

MASS FLOW MEASUREMENT OF MULTI-PHASE MIXTURES BY MEANS OF TOMOGRAPHIC TECHNIQUES

Gavin Teague

**Thesis Presented for the Degree of
DOCTOR OF PHILOSOPHY
in the Department of Electrical Engineering at the
UNIVERSITY OF CAPE TOWN
September 2002**

ACKNOWLEDGEMENTS

The author wishes to thank the following people for the assistance they provided over the duration of this thesis:

- Prof. J Tapson for his knowledgeable supervision and guidance over the past two years
- DebTech, and specifically A Roeb1, for the financial backing of the project and belief in its potential
- V Ramphal for the development of the MOSFET driver circuitry at the Cape Technikon
- B Teague for his excellent advice and assistance in the design and construction of the flow-simulation apparatus and other test equipment
- M Teague for the endless supply of nylon stockings used to construct the gravel masses.

TERMS OF REFERENCE

This project was initiated by A RoebI of DebTech (Control and Instrumentation) on 1 January 2001. The long-term goal of this research is to develop an instrument for the monitoring of a multi-phase airlift in an offshore mining application. Previous research had been done to assess how applicable impedance tomography is to the monitoring of an air-gravel-seawater mixture. However, the previous work has considered only static pipeline configurations. This research was initiated to assess the performance of impedance tomography for the on-line monitoring of a flow, and in particular, the calculation of the individual component mass flow rates within that flow. Consequently, it was necessary to assess the volume fraction prediction accuracy of the impedance tomography system, as well as the accuracy of the component velocities measured using the cross-correlation of a dual-plane system.

The specific instructions were to:

1. Verify the performance of an impedance tomography system on a laboratory-scale rig for static configurations of a multi-phase air-gravel-seawater mixture.
2. Construct two complete impedance tomography systems and interface them for flow measurement.
3. Develop the neural network software that accompanies the laboratory-scale rig for the volume fraction prediction of the phases within the mixture. Included in the software must be the calculation of the individual component velocities so that the mass flow rates of the phases within the mixture can be determined.
4. Perform a thorough characterisation of the dual-plane impedance tomography system.
5. Demonstrate the operation of the flow measurement system.
6. Draw conclusions regarding the application of a dual-plane impedance tomography system to the on-line monitoring of an air-gravel-seawater mixture.
7. Make recommendations for future project development and to discuss any remaining issues that would need to be addressed before an industrial prototype was developed.
8. Submit the results of the research and all supporting documentation by 30 June 2002.

In terms of support, a technical assistant from the Cape Technikon was allocated to this project to investigate a particular module of the system, namely the transmitter module. This will be discussed in greater detail in section 5.2.1 on page 41. Further, the services of an external electronic design company were employed for the printed circuit board design and manufacture. Finally, DebTech was involved in the design and construction of the laboratory-scale test rig used to assess the system's performance.

SYNOPSIS

This thesis investigates the use of a dual-plane impedance tomography system to calculate the individual mass flow rates of the components in an air-gravel-seawater mixture. The long-term goal of this research is to develop a multi-phase flowmeter for the on-line monitoring of an airlift used in an offshore mining application. This requires the measurement of both the individual component volume fractions and their velocities. Tomography provides a convenient non-intrusive technique to obtain this information.

Capacitance tomography is used to reconstruct the dielectric distribution of the material within a pipeline. It is based on the concept that the capacitance of a pair of electrodes depends on the dielectric distribution of the material between the electrodes. By mounting a number of electrodes around the periphery of the pipeline, and measuring the capacitances of the different electrode combinations, it is possible to reconstruct the distribution of the phases within the pipeline, provided the phases have different dielectric constants. Resistance tomography is used to reconstruct the resistivity distribution within the cross-section of the pipeline and operates in a similar way to capacitance tomography. Impedance tomography can be described as a dual-modal approach since both the capacitance and conductance of the different electrode combinations are measured to reconstruct the complex impedance of the material distribution.

Previous research has shown that impedance tomography can be used to reconstruct a three-phase air-gravel-water mixture [3, 4]. In addition, it has been shown that neural networks can be used to perform this reconstruction task [3, 4]. In particular, a single-layer feed-forward neural network with a 1-of-C output encoding can be trained to perform a three-phase image reconstruction. Further, a double-layer feed-forward neural network can be trained to predict the volume fractions of the three phases within the flow directly, based on the capacitance and conductance readings obtained from the data acquisition system. However, these tests were only for static configurations. This thesis will readdress this problem from the dynamic viewpoint. In addition, the individual component velocities will be calculated using the cross-correlation of the volume fraction predictions from two impedance tomography systems spaced a certain distance apart.

Cross-correlation flow measurement is based on the principle of 'tagging' disturbances in the flow generated through turbulence or suspended particles [12]. If the disturbance detected by the upstream sensor reappears at the downstream sensor after a period of τ seconds, and the separation between the two sensors is known to be L , then the velocity of the disturbance can be calculated as $v = L/\tau$. The time τ is calculated as the peak of the cross-correlation function. The present system is required to measure component velocities up to 20m/s. Based on this maximum velocity specification, it was determined that an on-line frame-rate of 200frames/s from both planes simultaneously is required for the intended application. However, a literature study revealed that none of the commercially available tomography systems could achieve this frame-rate. Further, no results had been published to-date of any research systems that could achieve a 200frames/s frame-rate for a dual-plane 16-electrode impedance tomography system. Consequently, it was necessary to design a new impedance tomography data capture technique to address this problem.

To understand the potential speed advantages of the new data capture technique, it is necessary to briefly examine standard tomography data acquisition systems. Essentially, standard tomography systems employ time division multiplexing (TDM). In other words, each electrode pair has a set time during which the corresponding capacitance

or conductance of that electrode pair is measured. The system consists of a single set of measurement electronics and the electrodes for a specific electrode pair are temporarily connected to the measurement electronics through multiplexed switches. These measurements are performed serially and therefore the time taken to capture a complete frame of data is the number of electrode combinations multiplied by the time taken to perform an individual measurement. Further, the process of switching can introduce impulses in the received signal, which increase the measurement time [37]. Subsequently, the time taken to collect a complete frame of data can become significant, thus limiting the on-line frame-rate of the system.

In contrast, the impedance tomography data acquisition technique developed for this application uses frequency division multiplexing (FDM), where the measurements are separated in the frequency domain. The operation of this technique will be explained for the simple configuration of a 4-electrode system, although it is completely scaleable and has been successfully implemented on a 16-electrode system. FDM impedance tomography systems specify electrodes as either dedicated transmitter or receiver electrodes. For the 4-electrode system, the order of these electrodes around the pipeline perimeter would be transmitter A, receiver A, transmitter B, receiver B. Each transmitter outputs a sine wave at a different frequency and all transmitters operate at the same time. Hence, receiver A will detect a signal from transmitter A modulated by the material between receiver A and transmitter A. Specifically, the conductance and capacitance of the material between this particular transmitter-receiver electrode pair modulates the magnitude and phase of this component. Further, receiver A will detect a signal from transmitter B, except that the material between receiver A and transmitter B will modulate this component. Since each transmitter operates at a different frequency, the received signal can be separated into the individual components using synchronous detection. Synchronous detection is also called phase-sensitive detection, lock-in detection and coherent demodulation. In this way, the capacitance and conductance of all the different electrode combinations are continuously available at the outputs of the synchronous detectors and no multiplexing is required. Subsequently, the potential frame-rate of the system is only limited by the bandwidth of the electronics.

The only drawback of the FDM impedance tomography system is a reduction in the number of electrode combinations for a given number of electrodes, compared to TDM systems. Specifically, only $\left(\frac{N}{2}\right)^2$ transmitter-receiver electrode pairs are obtained for an N-electrode system. To assess the consequences of this reduction in the number of electrode combinations, an 8-electrode TDM impedance tomography system was modified into a FDM impedance tomography system. Tests were conducted for static three-phase configurations and the reconstructions were performed using a volume fraction predictor neural network. The performance of the volume fraction predictor neural network was assessed on a database of test cases using the volume fraction error, which is the mean absolute error in the volume fraction predictions for each of the three phases. Image reconstructions were also performed but purely for verification purposes.

In a surprising result, the FDM impedance tomography system outperformed the TDM impedance tomography system for the air, gravel and water volume fraction predictions. Specifically, the air volume fraction error was reduced from 8.9% for the TDM impedance tomography system to 7.4% for the FDM impedance tomography system. All results represent the absolute error and are rounded off to two significant digits. Further, the gravel volume fraction error dropped from 5.0% to 4.7% and the water volume fraction error dropped from 6.5% to 4.8%. This improvement in performance is believed to be the result of the FDM impedance tomography system providing individual paths for the measurement of both the capacitance and conductance of the different electrode

combinations. Subsequently, each synchronous detector can be tuned individually for the optimum measurement range and gain of the corresponding transmitter-receiver electrode pair. This is in contrast to the TDM impedance tomography system where a single measurement system must be tuned as a compromise between all the electrode combinations.

Having verified the operation of the FDM impedance tomography data capture technique, it was necessary to construct a laboratory-scale rig for the assessment of the impedance tomography system on dynamic flow simulations. The diameter of the pipeline is 350mm. The laboratory-scale rig stands vertically and consists of a top section, two impedance tomography measurement sections and a bottom section. A flow-simulation apparatus was designed for the rig and is used to simulate the movement of air or gravel masses in the water at specified velocities under computer control. Each measurement section consists of a central plane of 16 capacitance measurement plates mounted on the outside of the pipeline. Driven axial guard electrodes are provided above and below this central plane of measurement electrodes. At the centre of each capacitance plate is a conductance probe mounted through the pipeline wall and in electrical contact with the water. Grounded radial shields pass between the measurement electrodes and an outer grounded shield prevents external interference.

Various modifications were made to the data acquisition system of the laboratory-scale rig based on the results of the 8-electrode system in an attempt to improve the volume fraction prediction accuracy. To assess the results of these modifications, the laboratory-scale measurement sections were tested on static configurations of air-gravel-water mixtures. A neural network was again used to perform the volume fraction prediction task and the volume fraction error for each of the three phases was assessed using a database of test cases. Specifically, the air volume fraction error was 1.2% for the top system and 1.4% for the bottom system, the gravel volume fraction error was 1.5% for the top system and 1.7% for the bottom system and the water volume fraction error was 0.86% for the top system and 1.0% for the bottom system. Clearly, the volume fraction errors of the 16-electrode impedance tomography systems were considerably smaller than the 8-electrode system, as expected. Further, tests revealed that the resolution of the system had improved to where an air bubble corresponding to 10% of the pipeline diameter could be detected at the centre of the pipeline. This bubble corresponds to an air phase volume fraction of 0.01.

Although these results were promising, it is the performance of the FDM impedance tomography system for real-time on-line monitoring of dynamic flow simulations that sets it apart from TDM impedance tomography systems. Specifically, tests revealed that the on-line data capture rate of the 16-electrode dual-plane impedance tomography system was 280frames/s from both measurement planes simultaneously, which clearly meets the required specification of 200frames/s. Further, the neural network software permitted the on-line reconstruction of these captured frames. The on-line data capture and reconstruction rate of a two-phase air-water image reconstruction was 280frames/s. For the more advanced three-phase air-gravel-water reconstruction, the on-line data capture and reconstruction rate reduced to 140frames/s for image reconstruction. However, since it is only the individual volume fractions that are required, the frame-rate can be increased to 180frames/s using the volume fraction predictor neural network.

Since the positions of the simulated bubbles are precisely controlled, it was possible to determine what the volume fraction of a specific phase should be at a certain time. Subsequently, a dynamic volume fraction error was calculated based on the mean absolute error between the measured volume fraction profile and the desired volume

fraction profile for a specific test. The average dynamic volume fraction error for the air phase was 1.0% at the top system and 0.89% at the bottom system using a double-layer volume fraction predictor neural network. The average dynamic volume fraction error for the gravel phase was 1.5% at the top system and 1.6% at the bottom system. Since the water was static, no attempt was made to assess the dynamic volume fraction error of the water phase. These results are comparable with those of the static tests thus proving that neural networks trained on static configurations can be used to perform reconstructions of dynamic flow simulations.

The volume fraction profiles of the air and gravel phases at the top and bottom systems were correlated to determine the individual component velocities. Specifically, tests were conducted for the simpler two-phase air-water reconstruction. Using a 0.01 volume fraction air bubble, the mean absolute error of the air phase velocity measurement was 0.13m/s at the maximum flow-simulation apparatus velocity of 4.2m/s. This corresponds to a velocity discrimination of 3.1%, which is comparable to the theoretical limit calculated as 1.5% for the current system. In terms of the three-phase air-gravel-water reconstruction, tests revealed that the system was able to calculate the individual velocities of the air and gravel phases provided the reconstruction algorithm performed a satisfactory separation between these two phases.

This research concluded that frequency division multiplexed impedance tomography provides a means of performing on-line real-time reconstructions at a far higher frame-rate than standard time division multiplexed impedance tomography systems. Subsequently, it enables tomography to be used in industrial applications where previously the limited frame-rates of tomography systems were a hindrance. Whether it is applicable to this particular application or not depends on the results of further research and testing.

Various modifications have been recommended which should improve the performance of the system. As an example, an 8-electrode sine wave system is currently being developed. Due to the extremely high parallelism in FDM impedance tomography, the component count increases exponentially as the number of electrodes increases. Subsequently, the current FDM impedance tomography system is based on square wave transmitter outputs instead of sine wave, since it is cheaper to perform the multiplication in the synchronous detectors for a square wave system. Although the square wave approach achieves good results, tests revealed that it is limited in terms of the resolution and accuracy of the reconstruction results, specifically for three-phase mixtures. This is because of the ripple on the outputs of the low-pass filters resulting from the harmonics of the square waves mixing in the multipliers. A sine wave system should address some of these problems since it is a simple task to select transmitter frequencies for a sine wave system to ensure that no frequency components are generated within the passband of the system.

TABLE OF CONTENTS

	PAGE
Acknowledgments	i
Terms of Reference	ii
Synopsis	iii
List of Illustrations	ix
1. INTRODUCTION	1
2. TIME DIVISION MULTIPLEXED IMPEDANCE TOMOGRAPHY	5
2.1 Standard Capacitance Tomography Measurement Systems	5
2.2 Standard Conductance Tomography Measurement Systems	7
3. IMPEDANCE TOMOGRAPHY RECONSTRUCTION ALGORITHMS	9
3.1 Standard Impedance Tomography Reconstruction Algorithms	9
3.2 Neural Network Reconstruction of Impedance Tomography Systems	11
3.2.1 Multi-layer perceptron fundamentals	11
3.2.2 Training multi-layer perceptrons	14
4. FREQUENCY DIVISION MULTIPLEXED IMPEDANCE TOMOGRAPHY OF AN 8-ELECTRODE SYSTEM	18
4.1 Frequency Division Multiplexed Impedance Tomography Concept	19
4.2 Design of an 8-electrode System to Verify the Frequency Division Multiplexed Impedance Tomography Concept	23
4.2.1 Generation of transmitter frequencies	24
4.2.2 Output transmitter details	26
4.2.3 Conductance and capacitance receiver details	27
4.2.4 Synchronous detection of receiver signals	28
4.2.5 Capture of capacitance and conductance readings and the reconstruction results achieved	30
4.2.6 Limitations of the current system	33
5. FREQUENCY DIVISION MULTIPLEXED IMPEDANCE TOMOGRAPHY OF A 16-ELECTRODE SYSTEM	34
5.1 Construction of a Laboratory-Scale Rig to Simulate a Pipeline Section	34
5.1.1 Reasoning behind certain rig dimensions	34
5.1.2 Design of transmitter and receiver electrodes and guarding	36
5.2 Electronic Design of a 16-electrode Frequency Division Multiplexed Impedance Tomography System	39
5.2.1 Output transmitter details	41
5.2.2 Conductance and capacitance receiver details	43
5.2.3 Synchronous detection of receiver signals	44
5.2.4 Population-based incremental learning fundamentals	51
5.2.5 Generation of transmitter frequencies	52
5.3 Software Design of a 16-electrode Frequency Division Multiplexed Impedance Tomography system	57
5.3.1 Data capture software description	57
5.3.2 Training and test database generation software description	61
5.3.3 Neural network reconstruction software description	63
6. STATIC SITUATION RESULTS FOR THE 16-ELECTRODE IMPEDANCE TOMOGRAPHY SYSTEM	66
6.1 Analysis of Capacitance and Conductance Measurement Drift	67
6.2 Training and Test Database Generation	69
6.3 Neural Network Reconstruction Results	73
6.3.1 Two-phase air-water reconstruction results	73
6.3.2 Three-phase air-gravel-water reconstruction results	75

7.	DYNAMIC SITUATION RESULTS FOR THE DUAL-PLANE 16-ELECTRODE IMPEDANCE TOMOGRAPHY CROSS-CORRELATION SYSTEM	78
7.1	Cross-correlation Flowmeter Fundamentals	78
7.2	Real-time Sampling and Reconstruction Software Description	82
7.2.1	Design of a flow-simulation apparatus	82
7.2.2	Verifying dynamically acquired data	85
7.2.3	Cross-plane interference problem	90
7.3	Two-phase Air-Water Reconstruction Results	93
7.3.1	Real-time volume fraction prediction and image reconstruction results	93
7.3.2	Reconstruction results for an air plug passing through the measurement section	96
7.3.3	Reconstruction results for a homogenous bubble flow	98
7.3.4	Cross-correlation velocity prediction results	101
7.4	Three-phase Air-Gravel-Water Reconstruction Results	103
7.4.1	Real-time volume fraction prediction and image reconstruction results	103
7.4.2	Cross-correlation velocity prediction results	105
7.4.3	Limitations of three-phase reconstruction algorithms	106
8.	CONCLUSIONS	107
9.	RECOMMENDATIONS FOR FUTURE DEVELOPMENT	110
10.	REMAINING SYSTEM ISSUES THAT STILL NEED TO BE ADDRESSED	111
	LIST OF REFERENCES	112
APPENDIX A	TABLE OF THE MANUFACTURING COSTS FOR 16-ELECTRODE FREQUENCY DIVISION MULTIPLEXED IMPEDANCE TOMOGRAPHY SYSTEM	116
APPENDIX B	CIRCUIT DIAGRAMS FOR 8-ELECTRODE FREQUENCY DIVISION MULTIPLEXED IMPEDANCE TOMOGRAPHY SYSTEM	117
APPENDIX C	CIRCUIT DIAGRAMS FOR 16-ELECTRODE FREQUENCY DIVISION MULTIPLEXED IMPEDANCE TOMOGRAPHY SYSTEM	122
APPENDIX D	TABLES OF THE CAPACITANCE AND CONDUCTANCE READINGS FOR 16-ELECTRODE FREQUENCY DIVISION MULTIPLEXED IMPEDANCE TOMOGRAPHY SYSTEM	129
APPENDIX E	MICROCONTROLLER PROGRAM CODE FOR 8-ELECTRODE FREQUENCY DIVISION MULTIPLEXED IMPEDANCE TOMOGRAPHY SYSTEM	130
APPENDIX F	MICROCONTROLLER PROGRAM CODE FOR 16-ELECTRODE FREQUENCY DIVISION MULTIPLEXED IMPEDANCE TOMOGRAPHY SYSTEM	134
APPENDIX G	FUNDAMENTAL FREQUENCIES AND HARMONICS FOR 8-ELECTRODE FREQUENCY DIVISION MULTIPLEXED IMPEDANCE TOMOGRAPHY SYSTEM	135
APPENDIX H	FUNDAMENTAL FREQUENCIES AND HARMONICS FOR 16-ELECTRODE FREQUENCY DIVISION MULTIPLEXED IMPEDANCE TOMOGRAPHY SYSTEM	136
APPENDIX I	AUTOMATIC CODE GENERATOR PROGRAM CODE	137
APPENDIX J	PBIL OPTIMISATION ALGORITHM AND SIMULATOR PROGRAM CODE	138
APPENDIX K	SAMPLER PROGRAM CODE	142
APPENDIX L	NEURAL NETWORK PROGRAM CODE	152
APPENDIX M	SCREEN CAPTURES OF SAMPLER PROGRAM	202
APPENDIX N	SCREEN CAPTURES OF NEURAL NETWORK PROGRAM	204
APPENDIX O	PAPER PUBLISHED IN MEASUREMENT SCIENCE AND TECHNOLOGY JOURNAL ON AUTHOR'S PREVIOUS RESEARCH	211

LIST OF ILLUSTRATIONS

	PAGE
FIGURES	
2.1 Diagram illustrating the first two stages of the standard capacitance tomography data collection protocol, showing for each stage, a cross-section of the pipeline and the capacitance plates mounted around the external perimeter of the pipeline.	5
2.2 Diagram of the 28 capacitance measurements in an 8-electrode standard capacitance tomography system.	6
2.3 Block diagram of the transducer circuitry for a standard AC-based capacitance tomography system, where each electrode is connected to its own transducer circuitry.	6
2.4 Diagram illustrating the four-electrode adjacent pair measurement protocol for resistance tomography on a pipeline cross-section, where the electrodes are embedded in the pipeline wall.	7
3.1 Diagram illustrating the use of a single-layer feed-forward neural network to perform a two-phase air-seawater image reconstruction of the pipeline contents.	12
3.2 Diagram illustrating the use of a single-layer feed-forward neural network with 1-of-C output encoding to perform a three-phase air-gravel-seawater image reconstruction.	13
3.3 Diagram illustrating the use of a double-layer feed-forward neural network to perform a three-phase air-gravel-seawater volume fraction prediction.	14
4.1 Diagram of a pipeline cross-section, where the linkages between the electrodes demonstrate the generation of the different measurement permutations using multiple transmitters and receivers operating in parallel with frequency division multiplexed impedance tomography.	19
4.2 Block diagram illustrating the concept of frequency division multiplexed impedance tomography of an 8-electrode system.	20
4.3 Block diagram showing the interactions between the transmitters, receivers and main synchronous detection board in the 8-electrode frequency division multiplexed impedance tomography system.	24
4.4 Output frequency generation for the 8-electrode frequency division multiplexed impedance tomography system using a PIC16F84 microcontroller.	25
4.5 Diagram of the capacitance and conductance receiver for the 8-electrode frequency division multiplexed impedance tomography system.	27
4.6 Synchronous detection of the receiver signal using a DG303 CMOS analogue switch to emulate multiplication by a reference waveform.	29
5.1 Overall rig dimensions for laboratory-scale multi-phase pipeline section.	34
5.2 Photograph of laboratory-scale multi-phase pipeline section.	35
5.3 Diagram illustrating the generation of a homogenous electric field between the capacitance measurement electrodes using driven axial guard electrodes.	36
5.4 Dimensions of the transmitter and receiver electrodes and guards.	37
5.5 Photograph of the pipeline from above with the 16 stainless steel conductance probes visible at the top and bottom measurement sections.	38
5.6 Diagram illustrating the outer earthed screen and the projected radial guards passing between the capacitance plates as a cross-section of the measurement section.	39
5.7 Block diagram illustrating the interactions between the different circuit boards in a 16-electrode	

	frequency division multiplexed impedance tomography system, where the number next to each line indicates the number of signals.	40
5.8	Block diagram of the output transmitter with push-pull power MOSFET output stage.	42
5.9	Operation of cross-conduction prevention circuitry in push-pull power MOSFET output stage.	42
5.10	Block diagram of capacitance receiver.	43
5.11	Block diagram of conductance receiver.	43
5.12	Block diagram of the synchronous detector circuit board showing only one complete channel, whereas each synchronous detector circuit board actually supports two complete channels.	45
5.13	Example of the square wave harmonics mixing in the multiplication stage of the synchronous detector for a 4-electrode system.	48
5.14	Output square wave generation with 0° and 90° references for the synchronous detection of the conductance and capacitance signals respectively.	53
5.15	Block diagram of frequency generator circuit board showing the eight PIC16F84A microcontrollers used to generate the eight independent transmitter frequencies and their quadrature waveforms.	54
5.16	Comparison between the frequency spectrum measured using a spectrum analyser and the spectral peaks predicted by the simulation software, which are plotted as square markers, for the transmitter B-receiver G electrode pair.	56
5.17	Block diagram of sample controller board.	58
5.18	Diagram illustrating the interactions between the calling application, sample unit and EDR device driver, where the arrows indicate the program execution required to capture a set of frames using streaming.	60
5.19	Screen capture of the database generation form.	62
5.20	Screen capture of the neural network parameter form.	64
6.1	Drift analysis of both capacitance and conductance readings for transmitter A-receiver A and transmitter C-receiver A electrode pairs over a 15-hour period.	67
6.2	Drift analysis of the capacitance between transmitter A and receiver A with the measurement section full of air.	68
6.3	Photograph of the 'bed-of-hooks' base plate and clear acrylic lid used as the bubble placement system for the generation of static tests.	70
6.4	Photograph of the three sizes of air and gravel bubbles corresponding to 30%, 20% and 10% of the pipeline diameter respectively.	71
6.5	Screen captures of test cases for the two-phase air-water image reconstruction of the top frequency division multiplexed impedance tomography system, where the desired outputs are on the right and the network predictions are on the left in each screen capture.	74
6.6	Screen captures of test cases for a three-phase air-gravel-water image reconstruction of the top frequency division multiplexed impedance tomography system, where the desired outputs are on the right and the network predictions are on the left of each screen capture.	76
7.1	Comparison between the computations required to determine the average velocity, velocity profile and velocity vector for a given frame-rate and at a specified minimum flow velocity.	81
7.2	Diagram of the flow-simulation apparatus with included detail of the motor mount and positioning base.	83
7.3	Photograph of the flow-simulation apparatus standing next to the laboratory-scale rig.	84

7.4	Photograph of the flow-simulation apparatus for the specific test of a 0.04 volume fraction gravel and a 0.04 volume fraction air bubble passing through the pipeline at the same time.	84
7.5	Screen capture of the real-time sampling form.	86
7.6	Screen capture of the dynamic verification form.	89
7.7	Comparison between the capacitance output voltages for the transmitter A-receiver A electrode pair of the top system when only the top system is operating and when both the top and bottom measurement planes are operating at the same time.	91
7.8	Plot of both desired and measured volume fraction profiles at the top and bottom measurement systems for a simulated 0.2m/s flow consisting of a 0.04 volume fraction air bubble and a 0.01 volume fraction air bubble.	93
7.9	Screen captures of a 0.04 volume fraction air bubble near the left-hand edge of the rig and a 0.01 volume fraction air bubble near the right-hand edge of the rig moving through the measurement planes at the same time.	95
7.10	Effect of the bubble length on the volume fraction measurement accuracy.	96
7.11	Plots of capacitance and conductance variations for receiver A as the water level is lowered past the top measurement plane.	97
7.12	Screen captures of the top measurement system image reconstruction as the water level is lowered past the conductance probes.	98
7.13	Photograph of the sparger used to generate a homogenous bubble flow in the laboratory-scale rig.	99
7.14	Plot of the air volume fraction measurements at the top and bottom measurement planes for a homogenous bubble flow.	99
7.15	Screen captures of a homogenous bubble flow where, for each screen capture, the top frame is the top measurement plane and the bottom frame is the bottom measurement plane.	100
7.16	Cross-correlation velocity measurement accuracy for the 0.01 volume fraction air bubble.	101
7.17	Plots of the volume fraction profiles at the top and bottom measurement planes for two 0.04 volume fraction air bubbles moving at 1m/s and 0.2m/s respectively.	102
7.18	Screen captures of a simulated 0.2m/s bubble flow, where the 0.04 volume fraction gravel bubble is situated near the left-hand edge of the rig and the 0.04 volume fraction air bubble is situated near the right-hand edge of the rig.	104
7.19	Plots of the air and gravel volume fraction profiles at the top and bottom measurement planes, where the velocity of the 0.04 volume fraction air bubble was 0.5m/s and the velocity of the 0.04 volume fraction gravel bubble was 0.2m/s.	105

TABLES

1.1	Required specifications of the flowmeter to be developed for the on-line monitoring of an air-gravel-seawater mixture in an offshore mining application.	3
3.1	Dielectric constants of the materials used during testing.	15
4.1	Output frequencies generated with a 4MHz crystal for the 8-electrode system.	26
4.2	LM675 specifications appropriate to this application.	26
4.3	Comparison between the performance of the frequency division multiplexed impedance tomography system and a standard time division multiplexed impedance tomography system for a two-phase air-seawater image reconstruction using a single-layer feed-forward neural network.	31

4.4	Comparison between the performance of the frequency division multiplexed impedance tomography system and a standard time division multiplexed impedance tomography system for a two-phase gravel-seawater image reconstruction using a single-layer feed-forward neural network.	31
4.5	Comparison between the performance of the frequency division multiplexed impedance tomography system and a standard time division multiplexed impedance tomography system for a three-phase air-gravel-seawater image reconstruction using a single-layer feed-forward neural network with a 1-of-C output encoding.	32
4.6	Comparison between the performance of the frequency division multiplexed impedance tomography system and a standard time division multiplexed impedance tomography system for a three-phase air-gravel-seawater volume fraction prediction using a double-layer feed-forward neural network.	32
5.1	Transmitter frequencies generated using two PIC16F84 microcontrollers with 4.43361MHz and 5MHz crystals respectively.	52
5.2	Transmitter frequencies selected using PBIL optimisation algorithm with the crystal frequencies and division factors required to generate these frequencies.	55
6.1	Comparison between the drift of the capacitance and conductance readings over a 15-hour period using different conductance probe materials.	69
6.2	Comparison between the performance of the 16-electrode frequency division multiplexed impedance tomography system and the 8-electrode prototype frequency division multiplexed impedance tomography system for a two-phase air-water image reconstruction using a single-layer feed-forward neural network.	73
6.3	Performance of the 16-electrode frequency division multiplexed impedance tomography system for a two-phase air-water volume fraction prediction using a single-layer feed-forward neural network.	74
6.4	Comparison between the performance of the 16-electrode frequency division multiplexed impedance tomography system and the 8-electrode prototype frequency division multiplexed impedance tomography system for a three-phase air-gravel-water image reconstruction using a single-layer feed-forward neural network.	75
6.5	Performance of the 16-electrode frequency division multiplexed impedance tomography system for a three-phase air-gravel-water volume fraction prediction using a single-layer feed-forward neural network.	77
6.6	Comparison between the performance of the 16-electrode frequency division multiplexed impedance tomography system and the 8-electrode prototype frequency division multiplexed impedance tomography system for a three-phase air-gravel-water volume fraction prediction using a double-layer feed-forward neural network.	77
7.1	Comparison between the average peak-to-peak output ripple of the capacitance and conductance readings for the different configurations tested.	92
7.2	Comparison between the average dynamic volume fraction errors for the 60mm, 90mm and 190mm bubble respectively.	96
7.3	Comparison between the average dynamic volume fraction errors of the different neural network reconstruction techniques considered for a three-phase air-gravel-water reconstruction.	103

CHAPTER 1

INTRODUCTION

This thesis investigates the use of a dual-plane impedance tomography system to calculate the mass flow rate of an air-gravel-seawater mixture. The long-term goal of this project is to develop an instrument for the monitoring of a multi-phase airlift used in an offshore mining application. This instrument is to be placed in-line and will be required to calculate the individual volume fractions of the seawater, gravel and air phases as they pass through it. In addition, it will be required to monitor the velocities of the individual components so that the mass flow rate of each component can be calculated. This information is invaluable for the on-line monitoring and control of such a process. Due to the potential advantages that such an instrument would possess, this project has progressed through a number of stages over roughly a four-year period. Firstly, research was done by Quinton Smit to determine whether capacitance tomography is a feasible technique for the monitoring of an air-gravel-seawater mixture [1, 2]. Secondly, research was done by the author on neural networks as a means of improving the accuracy of the reconstructed images [3, 4]. Work to-date has considered only static situations. This thesis will address the issue of monitoring a flow, and in particular, the measurement of the individual component flow rates.

Turbulent multi-phase flows are characteristic of many industrial processes. Although experimental observations and measurements of such processes are extremely complex, tomography has developed over the last decade as a reliable method of investigating such systems [5]. Process tomography techniques provide a novel means to visualise the internal dynamics of industrial processes, such as multi-component flows in pneumatic conveyors and the process of mixing or separation in plant vessels. Electrical tomography can be classified as electrical resistance tomography, electrical capacitance tomography or electro-magnetic tomography. The term 'electrical impedance tomography' has been used to describe systems where only the resistive component is measured, as well as systems where both the capacitance and conductance components are measured thus providing the complex impedance of the vessel contents. In this thesis, electrical impedance tomography will be used to describe the latter configuration.

Electrical capacitance tomography attempts to reconstruct the dielectric distribution within the cross-section of a vessel based on the capacitances measured between plates mounted on the wall of the vessel. Capacitance tomography is used for visualising the internal structure of various industrial processes involving non-conducting substances [6]. It operates on the principle that the capacitance of a pair of electrodes depends on the dielectric distribution of the material between the electrodes. By mounting several of these electrodes on the periphery of the vessel, it is possible to reconstruct an image of the vessel contents. Electrical resistance tomography attempts to reconstruct the resistivity distribution within the cross-section of a vessel based on conductance measurements between electrodes mounted in the wall of the vessel. It is intended for the imaging of multi-phase conductive mixtures.

In order to discriminate between more than two phases, a dual-modality tomography system is required [7]. Dual-modality tomography makes use of separate measurement modalities to produce two tomographic images representing the same interrogated area or object [8]. Since different modalities are used, the resulting complementary images represent two different object property variations over the interrogated area. In particular, to image a three-phase flow requires more than one modality [9]. An electrical impedance tomography system will be used to provide both capacitance and conductance information to discriminate between the air, gravel and seawater phases. The conductance distribution will give a good separation between seawater and the other phases. Since the conductance of the gravel and air phases is effectively the same, capacitance tomography will provide the distinction between these two phases based on their different dielectric constants.

Various flowmeters exist using different measurement techniques. These systems are typically flow regime dependent. Hence, assumptions must be made beforehand regarding the flow regime of the system to be monitored since the instruments themselves are not capable of determining local flow information [10]. If these assumptions are inaccurate, then the calculated flow rates are also inaccurate. Tomography provides a way of extracting this local flow information, such as volume fraction and component velocity [10]. In practice, the frame-rates of current tomography systems limit the application of this technique to situations where the fluid is moving relatively slowly [11]. The velocity of individual components will be measured using the cross-correlation of signals from the top and bottom impedance tomography systems. Cross-correlation flow measurement is based on the principle of 'tagging' disturbances in the flow generated through turbulence or suspended particles [12]. If the disturbance detected by the upstream sensor reappears at the downstream sensor after a period of τ seconds, and the separation between the two sensors is known to be L , then the velocity V can be calculated as $V = \frac{L}{\tau}$. Various assumptions are made when using this technique of flow measurement and these will be detailed at a later stage.

The primary use of multi-phase flow monitoring equipment is the quantification of the mass or volumetric flow rates of individual components within a flow [13]. Since direct mass flow measurement techniques are rare for two-phase flows and non-existent for three-phase flows, an inferential method is required [13]. Using the velocity distribution V_C of a specific component calculated with the cross-correlation algorithm and the volume fraction of that component α_C , the volumetric flow rate Q of a particular component can be determined as

$$Q = \int_A \alpha_C V_C dA \quad (1.1)$$

where the integration is performed over the cross-sectional area of the vessel A [10]. Combining the volumetric flow rate with the individual component density, the mass flow rate can be calculated [13]. This density information can be determined in advance for most applications [14].

The specific objectives of this research were to:

1. verify the performance of an impedance tomography system on a laboratory-scale rig for static configurations of a multi-phase air-gravel-seawater mixture.
2. combine two of these measurement sections and determine whether any interference or cross-coupling exists between the two systems when both are operating at the same time.
3. examine the ability of a dual-plane impedance tomography system to accurately measure the individual component mass flow rates in simulated flow situations. Specifically, the accuracy of component volume fraction measurements must be determined, as well as the range and accuracy of the component velocity measurements.
4. evaluate the ability of the system to differentiate between an air and gravel mass moving through the measurement section simultaneously and the corresponding effect on the component velocity measurement.
5. draw conclusions regarding the application of a dual-plane impedance tomography system to the on-line monitoring of an air-gravel-seawater mixture.
6. make recommendations for future project development and discuss any remaining issues that would need to be addressed before an industrial prototype is developed.

The required specifications for this multi-phase flowmeter are detailed in table 1.1.

TABLE 1.1 REQUIRED SPECIFICATIONS OF THE FLOWMETER TO BE DEVELOPED FOR THE ON-LINE MONITORING OF AN AIR-GRAVEL-SEAWATER MIXTURE IN AN OFFSHORE MINING APPLICATION.	
DESCRIPTION	SPECIFICATION
PIPELINE DIAMETER	350mm
MIXTURE VELOCITY	up to 20m/s
SOLIDS CONTENT	5 to 10%
AIR CONTENT	approximately 30%
CARRIER MEDIUM	seawater

Initially, two commercial impedance tomography systems were to be purchased. These commercial systems would then have been used to conduct the above tests. However, a literature study revealed that the commercially available tomography systems were unable to operate at the frame-rate necessary for the intended application. In addition, cost was a prohibitive factor. Consequently, the objectives of this thesis were expanded to include the development of a low-cost high frame-rate impedance tomography system. Appendix A on page 116 contains a table of the approximate costs involved in manufacturing the system discussed in this thesis. In comparison to commercially available systems, this new system clearly meets the requirement of being a low-cost system.

Typical electrical tomography systems consist of the sensors, data acquisition system and reconstruction computer [15]. As mentioned previously, research has been done by the author on the use of neural networks as an alternate reconstruction algorithm for an 8-electrode impedance tomography system [3, 4]. Although the results of this thesis rely heavily on the reconstruction stage, the neural networks developed previously will be used. In addition, the sensors were constructed according to standard design principles. Hence, this thesis concentrates on the data acquisition stage, and specifically, the development of a high-speed data acquisition system capable of operating at the frame-rate necessary for the intended application.

A laboratory-scale flowmeter is developed to prove the operation of the concept. Consequently, no effort has been made to ensure that the system can be used in an industrial environment, although recommendations are made for the construction of such an industrial instrument. In addition, the dynamic performance of the rig will only be tested on simulated air or gravel masses. Since the seawater will be static, no attempt will be made to measure the mass flow rate of the seawater component. A scaled-down version of this system could be tested in a closed-loop pump circuit at a later stage. A high-speed camera would permit verification of the measurement system results.

To gain a better understanding of how the new data acquisition system achieves a much higher frame-rate than standard tomography systems, it is necessary to first examine the operation of standard capacitance and resistance tomography systems and this is done in Chapter 2. Central to tomography systems is the reconstruction stage and Chapter 3 examines standard reconstruction techniques and highlights the major results of the author's previous research. The concept of the new data acquisition system will be explained in Chapter 4. In addition, the static tomography rig used in the previous project research will be modified to incorporate this new measurement technique. A comparison will be made between the results of this modified system and the original system. Chapter 5 will discuss the construction of the laboratory-scale dual-plane impedance tomography flowmeter. Static reconstruction results will be provided in Chapter 6 to verify the operation of the new system. Chapter 7 will examine the performance of this laboratory-scale rig in simulated flow situations. Finally, conclusions will be drawn and recommendations made regarding future project development. Chapter 10 will complete the thesis by briefly examining the remaining issues of the current system that would need to be addressed before an industrial prototype is developed.

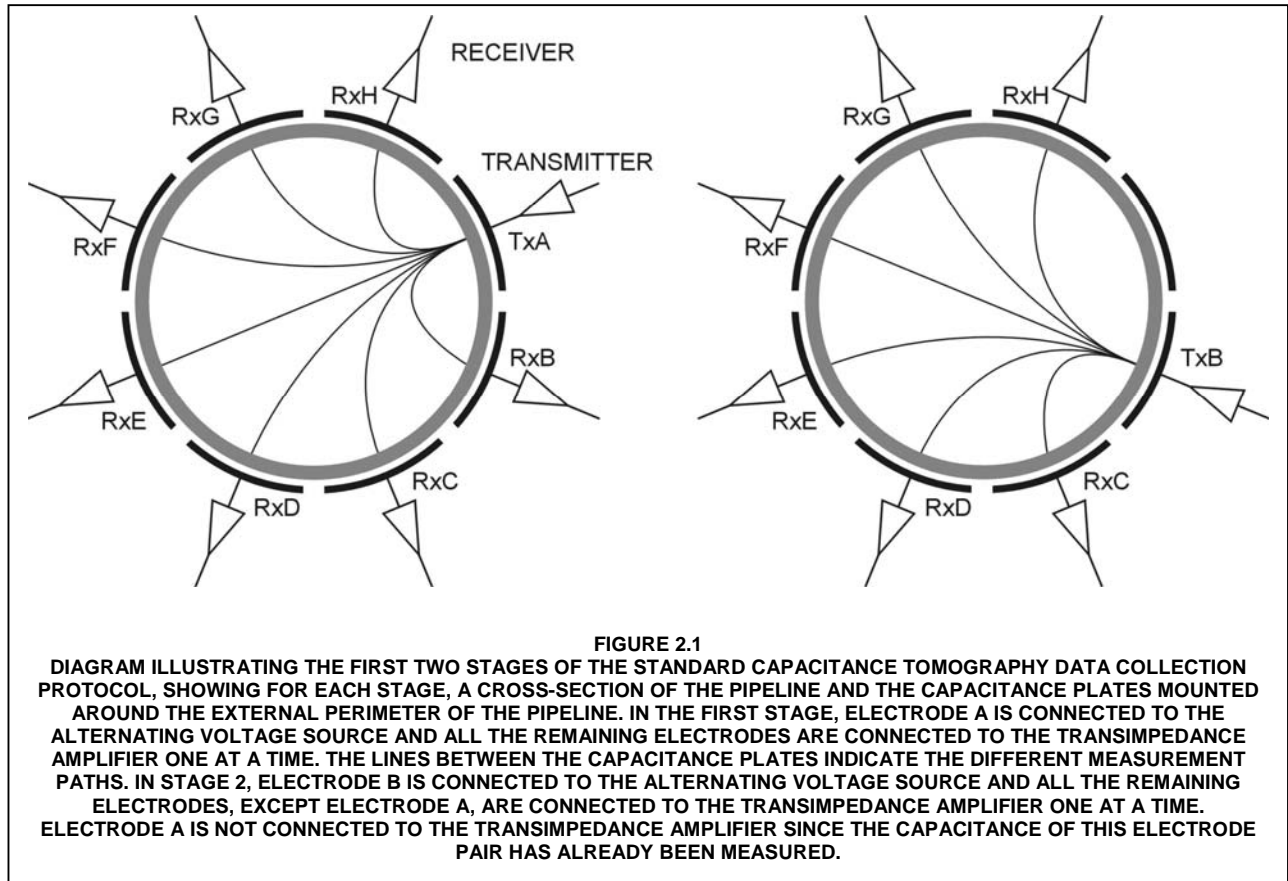
CHAPTER 2

TIME DIVISION MULTIPLEXED IMPEDANCE TOMOGRAPHY

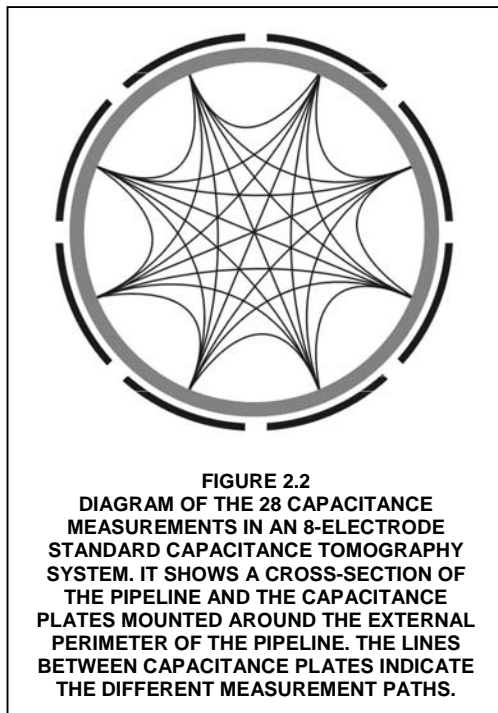
This chapter is a brief description of conventional capacitance and conductance tomography data acquisition systems. It will serve as a reference when the operation of the new data acquisition technique is explored in Chapter 4.

2.1 Standard Capacitance Tomography Measurement Systems

Capacitance tomography systems reconstruct the dielectric distribution of the pipeline cross-section. To achieve this, a number of capacitance plates are mounted around the measurement volume and the capacitances of all the different electrode combinations are measured. Various protocols define the order in which these capacitances are measured. The standard capacitance tomography data collection technique is referred to as protocol 1 where the measurements are made only between single pairs of electrodes. This measurement sequence involves applying an alternating voltage to the first electrode and then measuring the currents received on the remaining electrodes. The second electrode is then connected to this alternating voltage source and the currents are again measured on the remaining electrodes. The first two stages of this measurement protocol are illustrated in figure 2.1. Using this data collection protocol, the number of independent measurements obtained is $\frac{N(N-1)}{2}$ where N is the number of electrodes. Figure 2.2 on page 6 illustrates the 28 linearly independent capacitance measurements possible for an 8-electrode system.



Alternate data collection protocols do exist. For example, protocol 2 refers to grouping electrodes and exciting them in pairs. An advantage of these more complex data collection protocols is that a larger number of independent measurements are generated for the same number of electrodes, hence improving the resolution of the reconstructed images [11].

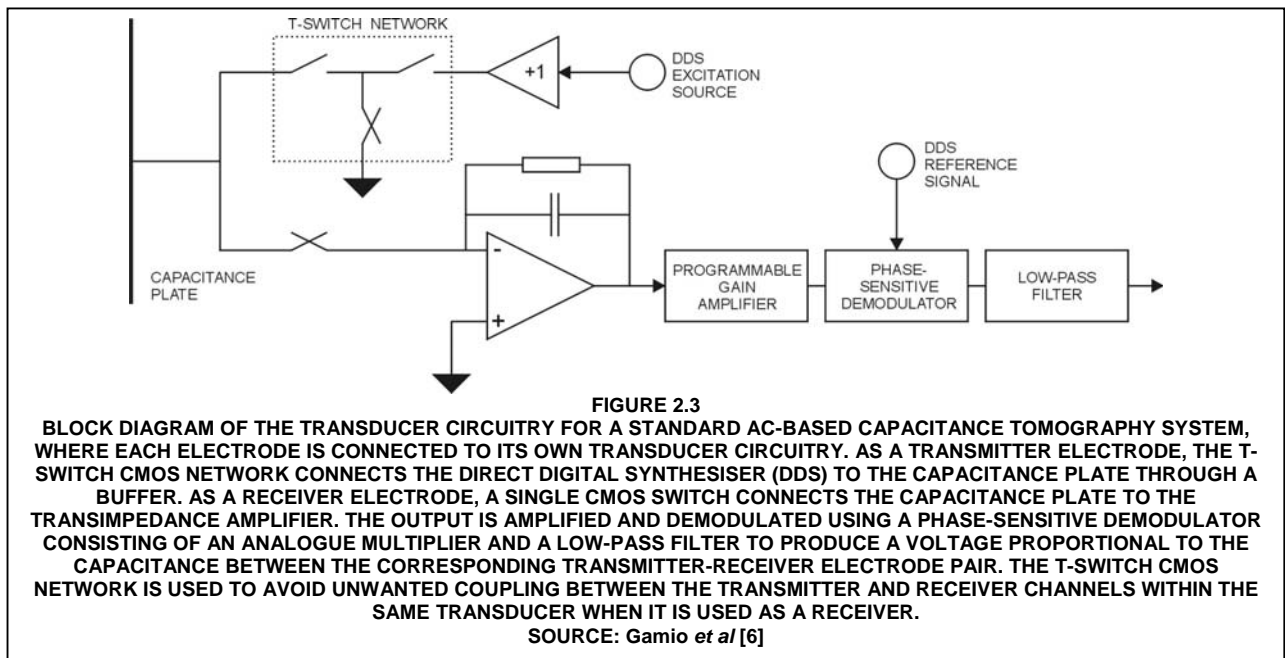


The capacitance of each transmitter-receiver electrode pair is measured using one of the following capacitance measurement techniques:

- Charge/discharge capacitance transducer
- AC-based capacitance transducer

It has been shown that the AC-based transducer has a high signal-to-noise ratio and good stray-immunity compared to the charge/discharge technique [6]. Stray immunity is the ability to measure small inter-electrode capacitances while ignoring the large stray capacitances to ground. Gamio *et al* describe a standard AC-based capacitance tomography system as follows [6]: a sinusoidal signal is generated by a direct digital synthesiser and buffered before being applied to the transmitter electrode through a CMOS switch network. The charge on the receiver electrode is measured by an op-amp with capacitive and resistive feedback, temporarily connected to the receiver electrode through a CMOS switch. A programmable gain amplifier amplifies the op-amp output before it is demodulated using a phase-sensitive

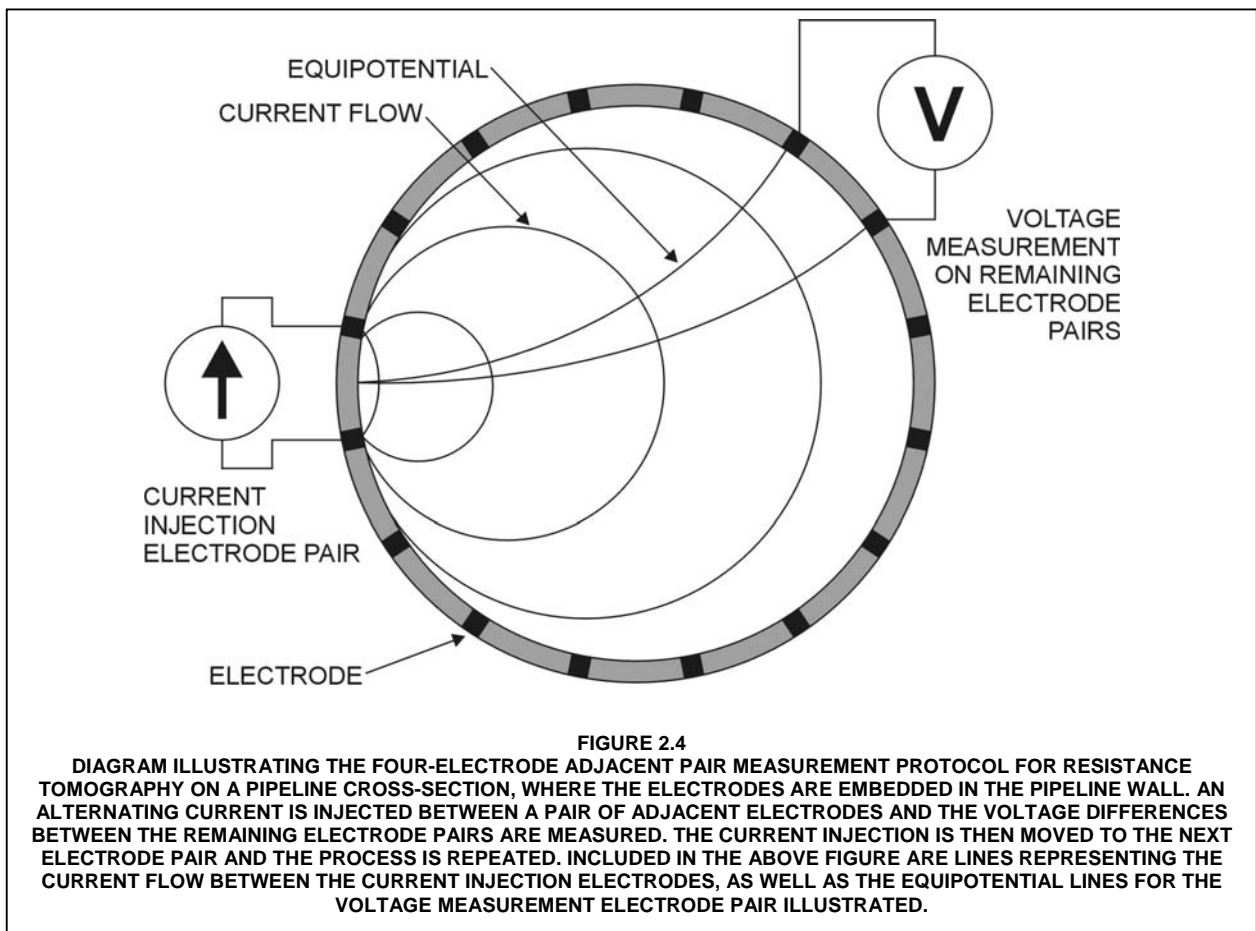
demodulator consisting of an analogue multiplier and low-pass filter. Figure 2.3 is a block diagram of this system.



The frame-rates of standard capacitance tomography data acquisition systems are fairly limited. Jaworski and Dyakowski made use of a dual-plane capacitance tomography system to monitor a pneumatic conveying system. The data acquisition rate of the 8-electrode capacitance tomography system was 100frames/s from both planes simultaneously [5]. Yang developed an AC-based 12-electrode capacitance tomography system using phase-sensitive demodulation. The frame-rate achieved was 140frames/s [16].

2.2 Standard Conductance Tomography Measurement Systems

Resistance tomography systems reconstruct the resistivity distribution of the pipeline cross-section. To achieve this, a number of conductance probes are mounted in the pipeline wall around the measurement volume. The standard data collection protocol for resistance tomography is the 'four-electrode adjacent pair measurement protocol'. It operates by injecting an alternating current from one electrode to another for an adjacent pair of electrodes. The voltages between all remaining adjacent electrode pairs are then measured [17]. The current injection is then moved to the next electrode pair and the process is repeated. To avoid contact impedance problems, the voltages of the electrodes adjacent to and including the injection pair are not measured. The total number of independent measurements obtained is $\frac{N(N-3)}{2}$ where N is the number of electrodes [18]. Figure 2.4 illustrates the four-electrode adjacent pair measurement protocol.



The motivation for the adjacent measurement protocol is to ensure that any variation in the contact impedance between the electrodes and the substance being imaged does not influence the measured voltages [17]. As a result of the forced current, various reactions may occur at the injection electrodes. At the voltage levels typically used, the predominant reaction is the electrolysis of water [19]. This consists of the oxidation of one of the injection electrodes and the reduction of the other injection electrode [19]. At the receiver, the input amplifiers typically have high input impedances, so minimal current flows between the ions in solution and the electrode. Consequently, no electrochemical reactions are present at the receiver electrodes [19]. Since minimal current flows into the receiver electrodes, the contact impedance between the electrode and solution as a result of electrode fouling only has a minimal effect.

The total net charge transported through the solution should be zero to ensure that its properties are not permanently changed [20]. The bi-directional current has the advantage of eliminating any long-term polarisation effects at the electrodes [19], depending on the conductance probe material [20]. The potential differences between the electrodes and solution that occur at an electrochemical interface may not all be the same resulting in voltage offsets. These offsets may be the result of concentration fluctuations in the solution, or variations in the electrode properties [20]. The problems associated with these offset voltages will be re-examined at a later stage.

The potential difference between adjacent electrodes is measured using a differential amplifier. Input and output multiplexers temporarily connect the current source to the injection electrodes and the differential amplifier to the detection electrodes. A phase-sensitive detector demodulates the differential amplifier output whose amplitude is modulated by the material within the vessel. A low-pass filter removes the switching harmonics to produce a DC voltage that is sampled by an analogue-to-digital converter [17].

Chapter 3 examines the reconstruction algorithms for tomography systems in greater detail.

CHAPTER 3

IMPEDANCE TOMOGRAPHY RECONSTRUCTION ALGORITHMS

It is the task of the reconstruction algorithm to determine an image of the contents of the vessel based on the limited capacitance and conductance measurements obtained from the data acquisition system [21]. The following section briefly discusses conventional reconstruction algorithms before neural network reconstruction techniques are discussed.

3.1 Standard Impedance Tomography Reconstruction Algorithms

The linear back-projection algorithm is common to both capacitance and resistance tomography systems. Fundamental to this technique is the generation beforehand of a sensitivity matrix. The image reconstruction is then performed in a single step through a matrix multiplication. For the capacitance tomography reconstruction, the sensitivity matrix describes how the capacitance readings for a particular electrode pair are affected by a change to the dielectric constant of a particular pixel within the vessel cross-section [11]. For resistance tomography, the sensitivity matrix, or Jacobian, maps the changes in the resistivity distribution to the changes in the voltages measured on the boundary of the region [22]. These matrices can be calculated numerically using the finite element modelling technique, or they can be measured experimentally [23].

It is actually the inverse of the sensitivity matrix that is required and the inverse problem that needs to be solved. For capacitance tomography, the inverse problem can be described as follows: given the electrode potentials applied to the vessel and a set of measured capacitances \mathbf{C} , calculate the permittivity distribution ϵ [24]. Since there are generally fewer readings than pixels in the reconstructed images, this inverse problem is ill-posed and some form of regularisation is required [25]. Since the sensitivity matrix is not square, an approximation technique is necessary to calculate the inverse. The standard linear back-projection algorithm simply uses the transpose of the sensitivity matrix [11]. The linearisation of the reconstruction imposes the following restrictions [25]:

1. there are no large changes in the conductivity or permittivity field
2. the spatial variation is low

Results produced using linear back-projection are typically blurry and are sub-optimal for a given set of capacitance or conductance readings. In addition, the second restriction results in images with very low spatial resolution [25].

If a better approximation to the inverse of the sensitivity matrix can be found, then better reconstruction results can be achieved. An example of an improved approximation technique is the single-step Tikhonov method [22]. The application of this method will be illustrated for the case of resistance tomography but is equally applicable to capacitance tomography. For electrical resistance tomography, the linearised forward problem is expressed using the following equation [22]

$$\mathbf{V} - \mathbf{V}_0 = \mathbf{J}_0(\rho - \rho_0) \quad (3.1)$$

where \mathbf{J}_0 is the Jacobian matrix that maps the change in resistivity values ρ to the change in voltages \mathbf{V} .

Since the change in voltages is actually what is observed and the change in resistivity is the desired output, a regularised solution to the inverse of the above equation is obtained as follows

$$\rho - \rho_0 = (J_0^T J_0 + \alpha A)^{-1} J_0^T (V - V_0) \quad (3.2)$$

where α is the regularisation parameter, A is typically the identity matrix and T indicates the transpose operation [22]. The matrix $(J_0^T J_0 + \alpha A)^{-1} J_0^T$ can be pre-calculated. Hence, approximate tomograms can be achieved very quickly simply through a matrix multiplication. The regularisation parameter α has a significant effect on the reconstructed images. If the regularisation is too large, then the images will be blurred and important features could be missed. If the regularisation is too small then the images are dominated by measurement noise and become nonsensical [22]. Various techniques exist to determine an optimal regularisation value.

Although simple to implement, linear back-projection is flawed and errors occur in the reconstruction resulting from the 'soft-field' nature of electrical sensors. A 'soft-field' sensor is one for which the contents of the vessel distort the measurement field. Consequently, the sensitivity of the sensor depends on the component distribution within the region of interest [26]. Various iterative reconstruction techniques exist as a solution to this problem.

Iterative reconstruction algorithms use the equation describing the forward problem to update the reconstructed images so that it converges on a more accurate solution. The process of iterative reconstruction algorithms, for the specific example of capacitance tomography, is described as follows [26]:

1. an initial image is calculated using the linear back-projection algorithm
2. the forward problem is solved for this approximate distribution using a finite element modelling technique and a set of capacitances are calculated for this approximate distribution
3. the error between the measured capacitances and these back-calculated values is used to calculate an error distribution
4. this error distribution is used to update the initial approximate distribution
5. this process is iterated until a satisfactorily small error between the measured and back-calculated capacitance readings is obtained

It is then necessary to check that the distribution did converge on a solution [11]. In particular, measurement noise can cause iterative image reconstruction algorithms to diverge [23]. In addition, multiple local optima are often encountered [25]. The algorithm can then become 'stuck' at these local optima [17]. Reconstruction time is also greatly increased as a result of the iterations making it unacceptable for industrial processes that have rapid dynamics [25]. According to Byars [11], iterative reconstruction techniques produce images of good resolution that are close to the theoretical limit determined by the measurement protocol and number of electrodes. An example of an iterative reconstruction algorithm for resistance tomography is the modified Newton-Raphson reconstruction algorithm.

The Newton-Raphson iteration is applied to the standard non-linear reconstruction algorithm, which minimises the mean square difference between the measured and estimated voltage responses as

$$\Phi(\rho) = (1/2)(f(\rho) - V_0)^T (f(\rho) - V_0) \quad (3.3)$$

where V_0 is the measured voltage and $f(\rho)$ is the estimated voltage for a resistivity distribution ρ [19]. Initially, a resistivity distribution ρ equal to the vessel filled with the background phase is assumed. The forward problem is then solved using a finite element model and a set of voltages is calculated for this resistivity distribution. These voltages are then compared to the measured voltages and the least-squares error is calculated between the two

voltages sets, as described above [27]. If it is less than a predefined value, then the reconstruction process is stopped. Otherwise, the resistivity distribution ρ is updated according to the following equation

$$\rho_{k+1} = \rho_k - \left[f'(\rho_k)^T f'(\rho_k) \right]^{-1} \left[f'(\rho_k)^T [f(\rho_k) - V_0] \right] \quad (3.4)$$

and the procedure is iterated until the convergence criterion has been met and a suitably accurate prediction obtained [19]. By incorporating prior knowledge of the structure of the matrices involved in this process, various modifications to the basic algorithm can be performed that reduce the total reconstruction time [28].

As mentioned previously, neural networks can be trained to perform image reconstructions of three-phase air-gravel-seawater mixtures [3, 4]. Although the training process may require a considerable amount of time to perform for the class of neural networks implemented, the network execution once the training is complete is fast since no iterations are required. Further, it was shown that a neural network could be used to perform on-line real-time reconstructions of an impedance tomography system and that the major source of delay in such an on-line process would be the data acquisition system, and not the reconstruction stage [3]. Since a high frame-rate is required for the intended application, neural network reconstruction techniques were employed.

3.2 Neural Network Reconstruction of Impedance Tomography Systems

A neural network is a highly parameterised mathematical function that offers a general framework for representing non-linear functional mappings from several input variables to several output variables [29]. Research into the use of neural networks for image reconstruction in capacitance tomography has been done [21, 24, 30, 31], although the previous work has predominantly made use of simulated capacitance readings obtained using a finite element modelling technique. The same is also true for resistance tomography systems [32, 33], whereas the author's previous research concentrated more on the practical application of neural networks to an existing impedance tomography system [3, 4]. Multi-layer perceptrons and radial basis function neural networks were analysed for the reconstruction task. Tests revealed that the multi-layer perceptron consistently out-performed the radial basis function neural network because of the limited amount of training data available. Hence, only the multi-layer perceptron was implemented in the new system software and will be discussed in the following section.

3.2.1 Multi-layer perceptron fundamentals

A multi-layer perceptron is a densely interconnected, layered network of neurons [29]. Each neuron consists of a combination function, which forms the weighted sum of the neuron inputs, and an activation function, which introduces a non-linearity into the network and confines the output to lie within a specific range.

To perform image reconstruction, each pixel in the image can be considered the output of an individual neural network performing a classification function for that particular pixel, namely

$$\text{pixel} = f_{\text{CLASSIFY}}(\vec{C}, \vec{G}) \quad (3.5)$$

where

$$\text{pixel} \in \{\text{air, gravel, seawater}\} \quad (3.6)$$

By combining these neural network outputs in parallel, an image of the contents of the vessel is obtained. If the algorithm proceeds to sum the air, gravel and seawater pixels in the image then an estimate of the individual component volume fractions within the vessel cross-section can be obtained.

Figure 3.1 illustrates a two-phase image reconstruction using a single-layer feed-forward neural network. The network consists of 56 input neurons corresponding to the 28 capacitance and 28 conductance readings obtained from the data acquisition system respectively. An additional input neuron is provided labelled 'offset' whose output is equal to unity. These input neurons are fully connected to the output layer of neurons. Although the image contains 100 pixels, only 88 of these are situated within the cross-section of the pipeline, hence there are 88 output neurons. Hyperbolic tangent activation functions are employed in the output layer. If the neuron output is less than zero then the corresponding pixel contains seawater, else it contains the second phase, which is air for the example illustrated.

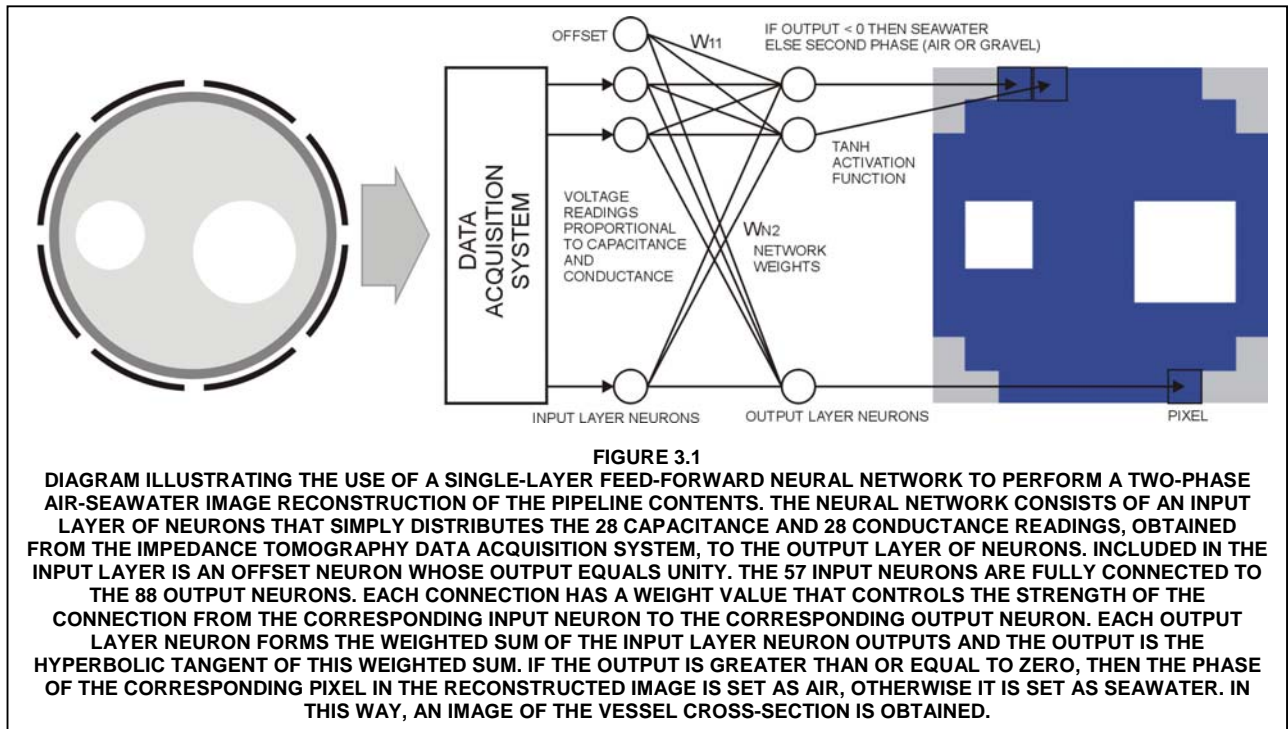
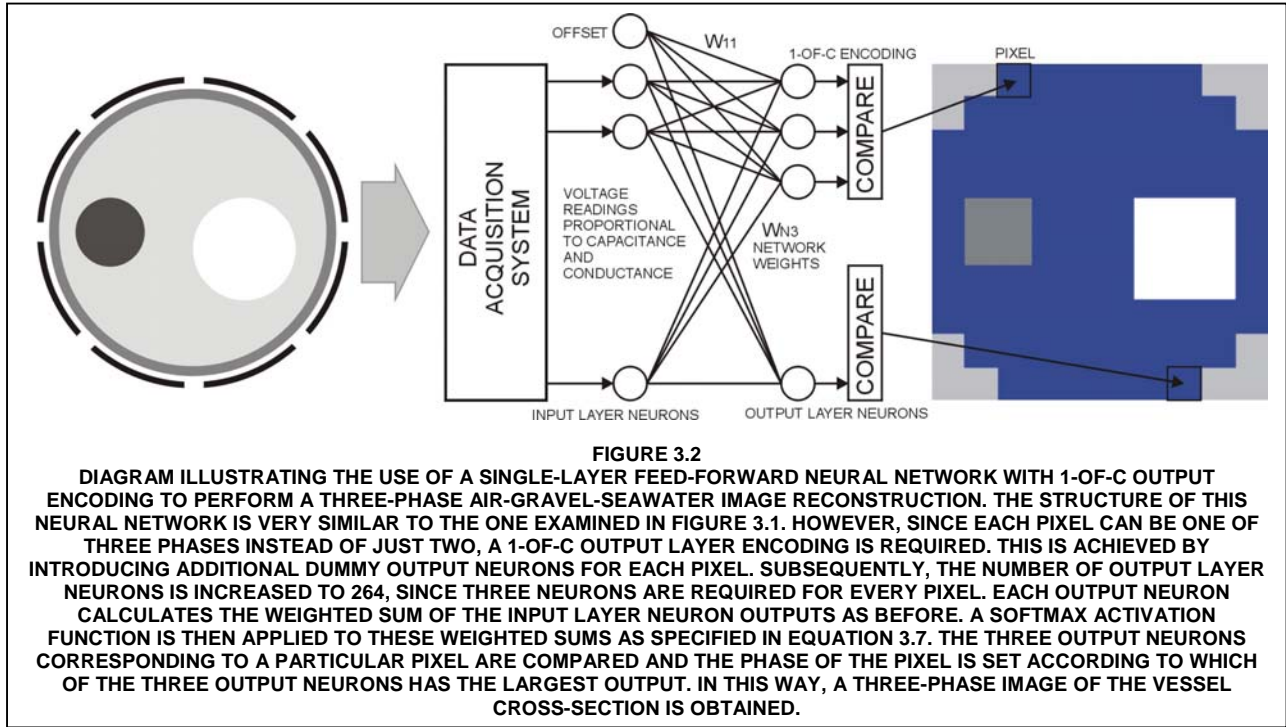


Figure 3.2 on page 13 illustrates a three-phase image reconstruction using a single-layer feed-forward neural network. The structure of this network is very similar to the two-phase image reconstruction neural network considered previously. However, since each pixel can be one of three phases instead of just two, a different output layer encoding is required. The standard multi-class classification scheme is 1-of-C encoding. A 1-of-C output encoding is achieved by introducing additional dummy output variables for each pixel. Subsequently, the number of output layer neurons is increased to 264, since three output neurons are required for every pixel. Each dummy output is given a target of zero except for that particular output corresponding to the desired classification, which has a target value of unity. In addition, each output neuron employs a softmax activation function [29]. This ensures that the sum of the three dummy output neurons always equals unity and so these outputs can be interpreted as valid posterior probabilities [29]. The pixel classification is the dummy output neuron with the largest output, and consequently the largest posterior probability.

The outputs are calculated using the following softmax activation function formula [29]

$$y_i = \frac{e^{a_i}}{\sum_{j=1}^3 e^{a_j}} \quad (3.7)$$

where a is the weighted input for each dummy output neuron. Although this neural network is able to perform three-phase air-gravel-seawater image reconstructions, the increase in the number of output neurons substantially increases the computation time of the reconstruction algorithm. This will be examined more closely when the issue of real-time on-line reconstructions is addressed.



Most tomography systems employ standard reconstruction algorithms to perform image reconstruction of the vessel contents. This image is then analysed to extract the required information for the interpretation or control of the corresponding process. However, improved results can typically be achieved by modifying the reconstruction algorithm to instead perform a parameterisation of the tomogram [25]. The parameters identified during reconstruction are exactly those required for interpretation or control. This has the advantage of more efficiently utilising the computational power available and reducing the reconstruction time [25]. Since the number of parameters is typically small compared to the number of pixels in the reconstructed images, the solution to the problem is also better posed than the standard image reconstruction task, and hence better results can be achieved.

Tests revealed that a neural network could be trained to predict the volume fractions directly [3, 4]. In this case, the network is required to model the functional mapping between the readings obtained from the data acquisition system and the continuous outputs representing the volume fractions of the air, gravel and seawater phases. This is achieved by reducing the number of output neurons to two, thus giving the volume fraction of gravel and air phases, respectively. The volume fraction of seawater is found by subtracting the sum of air and gravel volume fractions from 1. Secondly, linear activation functions are used for the two output neurons.

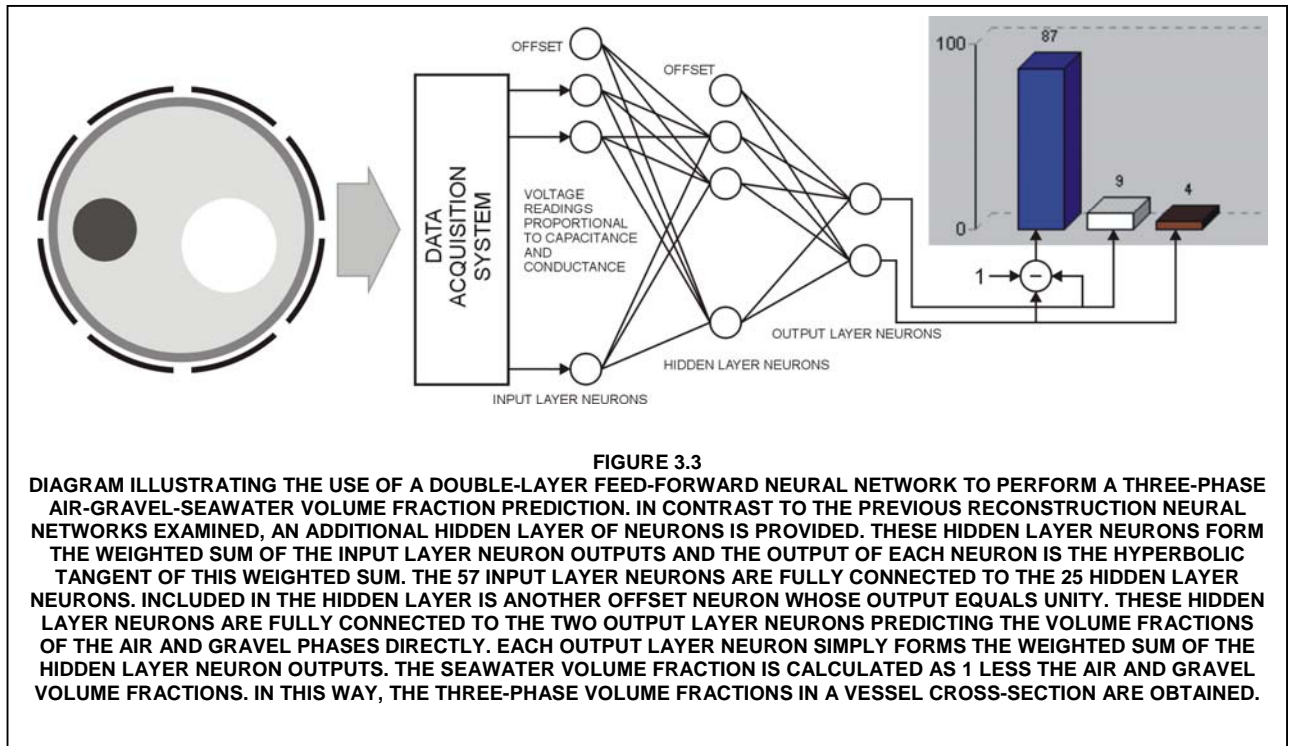
Consequently, the neural network performs a regression function, namely

$$[\text{gravel}_{\text{FRACTION}} \quad \text{air}_{\text{FRACTION}}] = f_{\text{REGRESSION}}(\bar{C}, \bar{G}) \quad (3.8)$$

where

$$\left. \begin{array}{l} \text{gravel}_{\text{FRACTION}} \\ \text{air}_{\text{FRACTION}} \end{array} \right\} \in [0,1] \quad (3.9)$$

Figure 3.3 illustrates the operation of a double-layer feed-forward neural network performing a volume fraction prediction. In contrast to the previous reconstruction neural networks examined, an additional hidden layer of neurons is provided. Specifically, tests concluded that the optimal number of hidden layer neurons for this application is 25. An offset neuron is also provided for the hidden layer. As before, the neural network has a fully connected structure between each neuron in the input layer and each neuron in the hidden layer, and between each neuron in the hidden layer and each of the two output neurons. These output neurons predict the air and gravel volume fractions directly. An additional advantage of predicting the volume fractions directly is a reduction in the computation time of the reconstruction.



3.2.2 Training multi-layer perceptrons

The adjustable parameters, or weights, of the network control the functional mapping and are obtained through network training. Training of a neural network is achieved through the presentation of known input-output pairs to the network and adjustment of the network weights until the error is satisfactorily small. To train a neural network requires a large quantity of representative data. Unlike previous research into neural network reconstruction techniques, the training data in this study was obtained from an existing impedance tomography rig. The generation of this large database required a method of systematically varying the vessel contents. A bubble placement system was developed for this task, consisting of a 'bed-of-nails' and clear acrylic lid. Figure 6.3 on page 70 is a photograph of a bubble placement system. It is used to locate a simulated air or gravel bubble at a

defined position within the vessel. The bubble placement system plays an important role in the training of a neural network. If the position of a bubble within the training database is not precisely defined, then the accuracy of the resulting neural network would be seriously degraded since the neural network would attempt to incorporate this outlier into the model of the system, subsequently compromising the performance for true data. Therefore great care must be taken during the generation of the training database to ensure that for each training case, the bubbles are positioned correctly.

It is important to note that the term ‘bubble’ does not refer to a bubble in the normal sense. Instead, it describes a contiguous mass of either gravel or polystyrene foam. Air bubbles were simulated as polystyrene foam cylinders since polystyrene foam has a dielectric constant that is similar to air, and is easier to work with. Gravel bubbles were simulated as collections of gravel chips. Figure 6.4 on page 71 is a photograph of the simulated air and gravel bubbles. Further, Table 3.1 lists the dielectric constants of the materials used during testing. Although the polystyrene entry in the table has a dielectric constant that is closer to gravel than air, it was polystyrene foam that was actually used, which has a dielectric constant similar to air.

PHASE	DIELECTRIC CONSTANT (AT 25°C)
AIR	1.00054
POLYSTYRENE	2.56
POLYSTYRENE FOAM	1.25
SAND – GRAVEL	3 to 5
WATER	78.54

The training database is generated as follows: essentially, the smallest identifiable element of one phase, or bubble, is placed in a specific location in the second phase using the bubble placement system. A set of readings is then taken corresponding to the capacitance and conductance measurements for this particular configuration. The bubble is then moved to a different position and a new set of readings is again taken and recorded. This process is repeated until the superposition of all the individual bubble positions effectively covers the vessel cross-section [21, 30, 31].

Drawn on the clear acrylic lid is a 10 by 10 matrix representing the pixels used to specify the desired network output. The user can identify which pixels are air, gravel or seawater simply by looking through the lid from above and setting the corresponding pixel values in the database generation software running on the reconstruction computer. This software also records the capacitance and conductance readings corresponding to this particular bubble location, and this database is used for training the neural network.

Neural networks are capable of generalisation for previously unseen bubble configurations. Generalisation is described as the ability to predict the correct outputs for new inputs even though they do not form part of the training database [29]. To test the generalisation performance of the neural network, a validation database was generated. This consists of combinations of air bubbles, combinations of gravel bubbles and combinations of both air and gravel bubbles, which do not form part of the training database. The validation database is used to determine whether the neural network is able to generalise from a single bubble of only one phase in the training database to combinations of bubbles of different phases, as would typically be achieved in a more realistic situation.

The generalisation ability of the neural network is assessed using the following performance measures:

1. Threshold error, which is the percentage of pixels that are not predicted correctly. For example, if the network predicts that a particular pixel is gravel when it is actually air, then this represents an error. These errors are summed over the entire database and are divided by the total number of pixels in the database. This performance measure applies only to the image reconstruction neural network and gives an indication of the positional correlation between the desired network output and the network prediction.
2. Volume fraction error, which is the percentage error of the volume fraction predictions for each of the three phases. For example, if the image reconstruction neural network predicts that there are 42 pixels of seawater when there are actually 44, then this represents an error of two pixels. These errors in the seawater prediction are summed over the entire database and are divided by the total number of pixels in the database to give the seawater volume fraction error. For the volume fraction predictor neural networks, the absolute values of the differences between the desired and the predicted volume fractions are summed over the entire database and then divided by the total number of frames in the database. It therefore represents the mean absolute error of the volume fraction predictor neural network for a specific phase. This measure is independent of pixel position and operates simply in terms of volume. It gives an indication of the ability of the network to predict volume fractions accurately.

The generalisation capability of the neural network is regularly monitored during network training. Early stopping is used to prevent the neural network from fitting the training data exactly, which would result in poor reconstruction results for unseen bubble configurations. The network training is stopped as soon as the generalisation performance starts to consistently deteriorate.

The neural network training algorithms implemented in this system will be discussed briefly in the following section for completeness. Training a multi-layer perceptron is typically performed in two stages. Firstly, the derivative of the error function with respect to the network weights must be determined. The standard sum-of-squares error function is as follows [29]

$$E = \frac{1}{2} \sum_{k=1}^C (y_k - t_k)^2 \quad (3.10)$$

where C is the total number of network outputs, y_k is the network prediction for output k and t_k is the desired prediction for output k . The derivative of this error function is calculated in a computationally efficient way using the back-propagation algorithm [29]. For the output layer of neurons, the derivative of the error function with respect to the network weights is calculated as [29]

$$\frac{\partial E}{\partial w_{ki}} = g'(a_k)(y_k - t_k)z_i \quad (3.11)$$

where $g'(a_k)$ is the derivative of the output neuron activation function for combined input a_k and z_i is the output of hidden layer neuron i with weight w_{ki} connecting to neuron k in the output layer. For the hidden layer neurons, the task of calculating the error derivative is complicated by the fact that there is no defined 'target' value. Although the accompanying software includes hidden layer capabilities, these will not be discussed here since an explanation of the hidden layer calculations requires a thorough analysis of the back-propagation technique. For those wishing to read further, Bishop provides an in-depth analysis of the back-propagation technique in [29 pp. 140-148].

Secondly, these computed derivatives are used to adjust the network weights so as to minimise the error function. Gradient descent and Resilient back-propagation were implemented in this system to perform this particular task. Gradient descent is a process whereby the gradient information is used to adjust the weights so as to move a small distance in the weight space in the direction in which the error function decreases most rapidly [29]. This is achieved using the following weight update equation [29]

$$w_{ij}(t+1) = w_{ij}(t) - \eta \frac{\partial E}{\partial w_{ij}} + \mu(w_{ij}(t) - w_{ij}(t-1)) \quad (3.12)$$

where η is the learning rate and controls the speed of convergence and μ is the momentum parameter that adds inertia to the movement of the algorithm through the weight space [29]. Resilient back-propagation is a local, adaptive learning algorithm that uses local gradient information to modify the weights of the network directly [34, 35, 36]. Unlike gradient descent, Resilient back-propagation simply makes use of the sign of the derivative and not the magnitude to update the network weights according to the following rule [36]

$$w_{ij}(t+1) = w_{ij}(t) + \begin{cases} -\Delta_{ij}(t) & \text{if } \frac{\partial E}{\partial w_{ij}} > 0 \\ \Delta_{ij}(t) & \text{if } \frac{\partial E}{\partial w_{ij}} < 0 \\ 0 & \text{otherwise} \end{cases} \quad (3.13)$$

where $\Delta_{ij}(t)$ is the weight update value for weight $w_{ij}(t)$ at epoch t . These weight update values are modified during the training process to give optimal convergence on the minimum of the error function.

The network training process can be summarised as follows. Initially the weights of the network are randomly set. The training data inputs are propagated through the network and the network outputs are calculated using these initial weights. The network predictions are then compared to the desired network outputs and the derivative of the error function is calculated using back-propagation. This gradient information is then used to adjust the network weights, using either gradient descent or Resilient back-propagation, and the process is repeated. Through multiple iterations, the error in the network predictions steadily decreases as the neural network learns to model the desired functional mapping inherent in the training database.

CHAPTER 4

FREQUENCY DIVISION MULTIPLEXED IMPEDANCE TOMOGRAPHY OF AN 8-ELECTRODE SYSTEM

In order to determine the velocity of components by cross-correlating the data from two planes, the real-time performance of the system must be guaranteed. This incorporates the image reconstruction and data processing tasks [12]. It can be shown that the frame-rate ν must meet the following requirement

$$\nu = \frac{1}{\left(\frac{2L}{V_{MAX}} + \frac{\Delta V}{V_{MAX}} \right)} \quad (4.1)$$

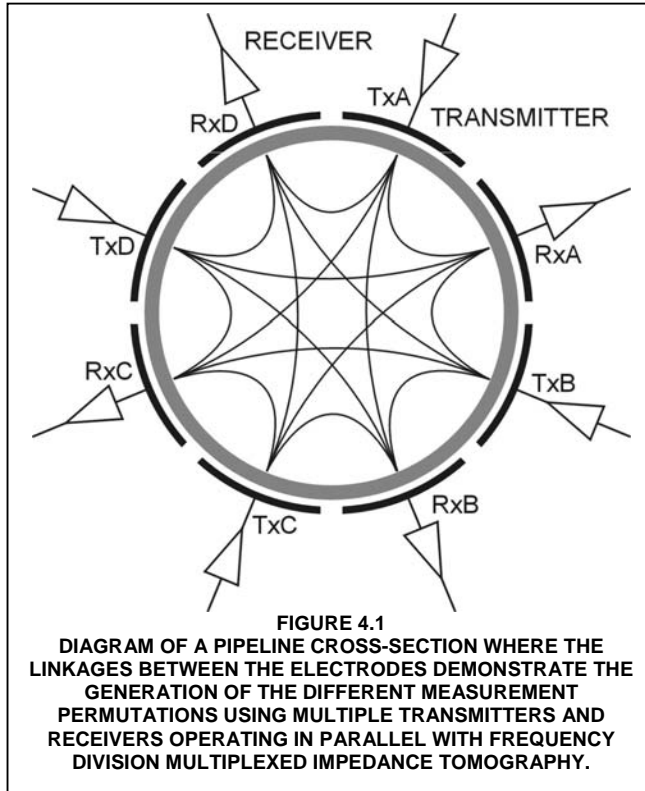
where L is the separation between the two planes, V_{MAX} is the maximum expected velocity and $\Delta V/V_{MAX}$ is the desired velocity discrimination [12]. A frame-rate of 200frames/s is required for a velocity discrimination of 10% with a maximum velocity of 20m/s and a system separation of approximately 500mm. No results had been published to-date of a 16-electrode impedance tomography system capable of operating on-line at this frame-rate. While the standard tomography measurement technique could possibly be pushed to a frame-rate close to that required, it is fundamentally limited by its sequential processing. Consequently, it was decided that the data acquisition problem should be examined from a different perspective to determine whether a different approach may yield higher frame-rates more easily.

Standard tomography systems are based on the concept of switching between different electrode combinations. Specifically, standard tomography systems can be described as using time division multiplexing (TDM) where the readings are performed serially and each reading receives a portion of time within which to measure the capacitance or conductance of a particular electrode pair. However, it is this switching and the transients that result from the switching that increases the overall time of measurement [37]. The measured signal passes through a multiplexing stage consisting of CMOS switches. As the signal is switched from one electrode to another, this switching action may produce a step-input to the next stage of processing. This signal experiences a long transit time through the output stage low-pass filter and consequently, the system must be held in this configuration for a longer time to get a stable reading [12]. Only once the output reading is stable, can the system then proceed to measure the properties of the next electrode pair. This greatly increases the total data acquisition time, thus limiting the real-time applicability of this technique. Deng *et al* made use of a zero-crossing switch to eliminate the negative effect of this impulse. Using this technique, a frame-rate of 50frames/s was achieved for a dual-plane 12-electrode resistance tomography system [12].

Within standard capacitance and resistance tomography systems, there does exist the possibility of parallelism. For example, in resistance tomography systems, the voltage differences for all the receiver electrode pairs corresponding to a particular current injection pair can be measured simultaneously simply by replicating the receiver circuitry [27]. This reduces the data acquisition times and removes the phase shift associated with additional multiplexing stages [27]. By using parallelism in the measurement, it has been shown that higher frame-rates can be achieved. Frequency division multiplexing can be considered as an extension of the use of parallelism to achieve extremely high frame-rates.

4.1 Frequency Division Multiplexed Impedance Tomography Concept

As previously mentioned, the use of parallelism in impedance tomography greatly increases the potential frame-rate of the system. Frequency division multiplexed (FDM) impedance tomography makes use of this concept to achieve extremely high frame-rates. Each electrode is either a dedicated transmitter or receiver electrode. Specifically, every second electrode is a transmitter electrode with the electrodes in between being receiver electrodes. This is illustrated in figure 4.1 for the case of an 8-electrode system. Included in figure 4.1 are linkages showing the possible measurement permutations for this configuration.



All the transmitters operate simultaneously.

Consequently, each receiver will receive signals from all the transmitters simultaneously. For example, receiver A will receive a signal from transmitter A modulated by the material between receiver A and transmitter A. It will also receive a signal from transmitter B except that this signal will be modulated by the material between transmitter B and receiver A. It will receive a signal from transmitter C modulated by the material between transmitter C and receiver A. Finally, it will receive a signal from transmitter D modulated by the material between transmitter D and receiver A. This configuration achieves maximum parallelism since all measurements can be made simultaneously without having to switch between different electrode configurations. The only remaining issue that needs to be addressed is how to separate the signals from the different transmitters, which are received simultaneously at a receiver.

In contrast to TDM impedance tomography systems, FDM impedance tomography separates the different signals in the frequency domain. Specifically, each transmitter operates at a different frequency in figure 4.1 above. The signal at a receiver output is therefore a superposition of these frequency components, where the amplitude and phase of a particular component has been modulated by the conductance and capacitance of the material between that particular transmitter-receiver electrode pair. Synchronous detection is then performed on the receiver output to separate this signal into the components from the different transmitters. Synchronous detection is also called phase-sensitive detection, lock-in detection and coherent demodulation. The following section will examine the practical application of this concept to an 8-electrode system. This concept will then be proven from a theoretical standpoint.

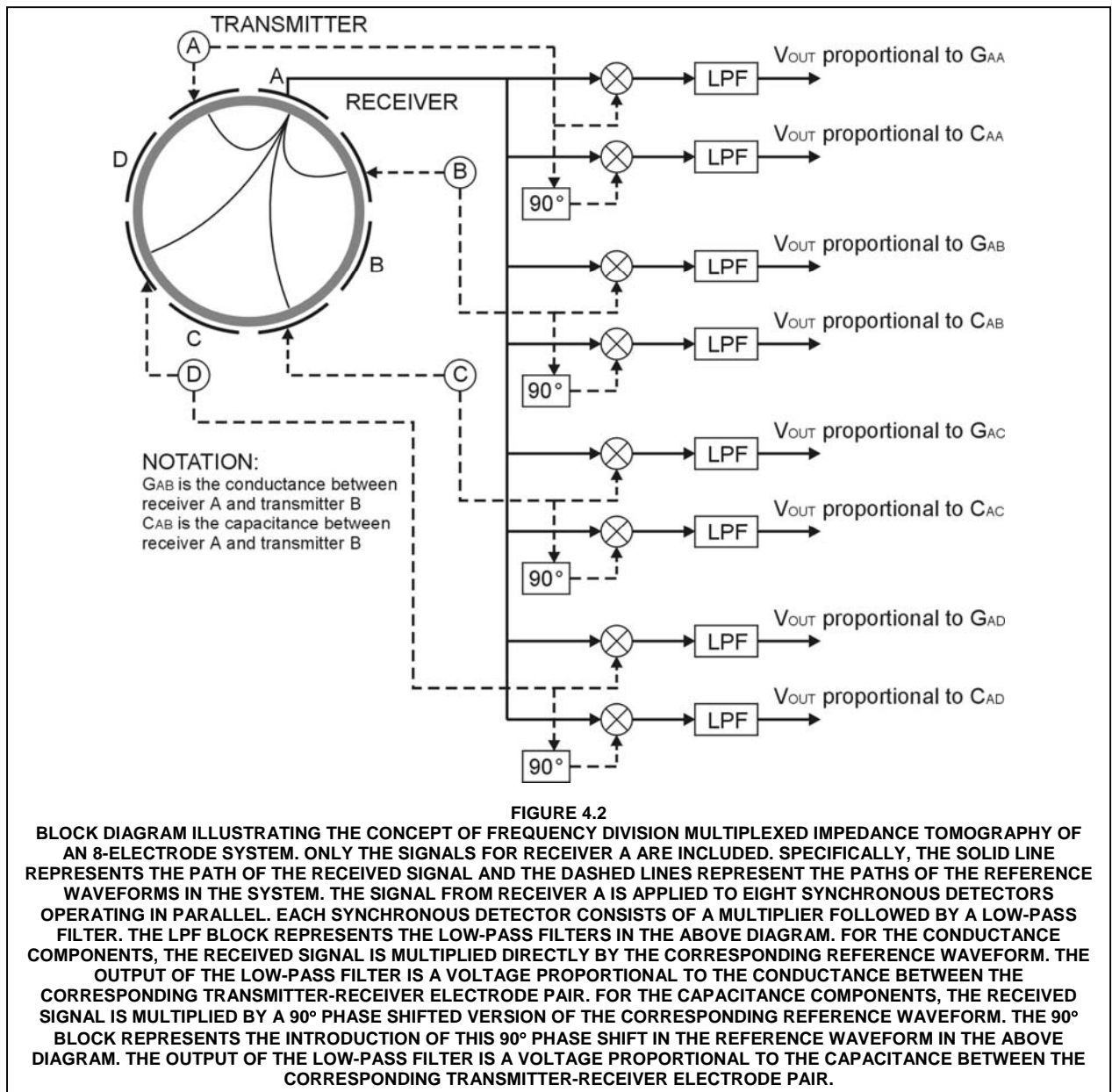


Figure 4.2 illustrates the application of this concept to an 8-electrode system. Specifically, only the signal from receiver A will be examined although it is a simple extension to perform the same analysis for the other receivers. As mentioned previously, the output from receiver A consists of a superposition of the signals from transmitter A, B, C and D. This signal is connected to eight synchronous detectors in parallel. For example, the first synchronous detector multiplies the received signal by the reference for transmitter A. After low-pass filtering, a DC voltage is produced whose magnitude is proportional to the conductance between receiver A and transmitter A. The second synchronous detector multiplies the received signal by a 90° phase shifted version of the reference for transmitter A. After low-pass filtering, a DC voltage is produced whose magnitude is proportional to the capacitance between receiver A and transmitter A. The third synchronous detector multiplies the received signal by the reference for transmitter B. After low-pass filtering, a DC voltage is produced proportional to the conductance between receiver A and transmitter B. This process is replicated for the remaining transmitters to measure both the capacitance and conductance between receiver A and each of the four transmitters. All this electronic circuitry is then replicated for receiver B, C and D.

Voltages proportional to the capacitance and conductance of all the different transmitter-receiver electrode pairs are continuously available on the outputs of the low-pass filters and no switching is required. In addition, since each transmitter-receiver electrode pair has its own detection path, each signal can be amplified individually for optimum dynamic range and offset. This is in contrast to standard tomography systems where one set of electronics is used to perform all the measurements and consequently must be tuned to some general setting that is a compromise for all electrode pairs. Since the readings are continuously available, the frame-rate of this system is only limited by the cutoff frequency of the low-pass filters. Subsequently, this technique can achieve far higher frame-rates than even the most advanced TDM impedance tomography systems.

One of the drawbacks of the FDM impedance tomography technique is that fewer measurements are available for a given number of electrodes, compared to the standard capacitance and conductance tomography protocols.

Specifically, the number of transmitter-receiver electrode pairs is $\left(\frac{N}{2}\right)^2$ where N is the number of electrodes.

Hence, for an 8-electrode impedance tomography system, only 16 capacitance and 16 conductance readings are available instead of 28 capacitance and 20 conductance readings. Consequently, a reduction in the resolution of the reconstructed images is expected. To achieve the same level of resolution it is therefore necessary to increase the number of electrodes.

Further, since separate transmitter and receiver electrodes are employed, the sensitivity of the FDM system is effectively half that of the corresponding TDM system. However, as mentioned previously, the use of separate transmitter and receiver electrodes ensures that no switching is required permitting extremely high frame-rates to be achieved. It is therefore a compromise between sensitivity and on-line frame-rate capability. This compromise will have to be made at various other stages during project design as well.

Another drawback of the large parallelism associated with FDM impedance tomography is an exponential increase in cost as the number of electrodes increases. For example, it was fairly cost-effective to develop the 8-electrode prototype system. On increasing the number of electrodes to 16, the cost became considerably more prohibitive and the circuit board area increased by a factor of 8. Appendix A on page 116 includes a table of the costs involved in manufacturing a dual-plane 16-electrode FDM impedance tomography system.

From a theoretical standpoint, the operation of the FDM impedance tomography concept can be considered for the simple case of a 4-electrode system with sine wave excitation. Given that the two transmitter outputs are described by the following equations

$$v_{TxA}(t) = \sin(w_A t) \quad (4.2)$$

$$v_{TxB}(t) = \sin(w_B t) \quad (4.3)$$

then the receiver output consists of a superposition of these two waveforms, where each waveform is scaled and phase shifted by the conductance and capacitance of that particular transmitter-receiver electrode pair. The output waveform for receiver A is therefore

$$v_{RxA}(t) = K_{TxARxA} \sin(w_A t + \phi_{TxARxA}) + K_{TxBRxA} \sin(w_B t + \phi_{TxBRxA}) \quad (4.4)$$

Each received signal can be resolved into two components, namely a conductance component that is in phase with the original transmitter output and a capacitance component that is phase shifted by 90° [38 pp. 30-35].

Consequently, the above equation can be rewritten as follows

$$v_{RxA}(t) = G_{TxARxA} \sin(w_A t) + C_{TxARxA} \cos(w_A t) + G_{TxBRxA} \sin(w_B t) + C_{TxBRxA} \cos(w_B t) \quad (4.5)$$

To determine the conductance between transmitter A and receiver A this signal is then multiplied by the reference for transmitter A as follows

$$v_{GMULTA}(t) = G_{TxARxA} \sin(w_A t) \sin(w_A t) + C_{TxARxA} \cos(w_A t) \sin(w_A t) + G_{TxBRxA} \sin(w_B t) \sin(w_A t) + C_{TxBRxA} \cos(w_B t) \sin(w_A t) \quad (4.6)$$

which can be rewritten using trigonometric identities as

$$v_{GMULTA}(t) = \frac{G_{TxARxA}}{2} [1 - \cos(2w_A t)] + \frac{C_{TxARxA}}{2} [\sin(2w_A t)] + \frac{G_{TxBRxA}}{2} [\cos(w_B t - w_A t) - \cos(w_B t + w_A t)] + \frac{C_{TxBRxA}}{2} [\sin(w_B t + w_A t) - \sin(w_B t - w_A t)] \quad (4.7)$$

$$v_{GMULTA}(t) = \frac{G_{TxARxA}}{2} \quad \text{if } w_{LPF} < \min(2w_A, w_B + w_A, |w_B - w_A|) \quad (4.8)$$

Consequently, the multiplier output is proportional to the conductance between transmitter A and receiver A provided the cutoff frequency of the low-pass filter is less than the minimum of $2w_A$, $w_B - w_A$ or $w_B + w_A$.

To determine the conductance between transmitter B and receiver A the received signal is multiplied by the reference for transmitter B and the output is as follows

$$v_{GMULTB}(t) = \frac{G_{TxARxA}}{2} [\cos(w_A t - w_B t) - \cos(w_A t + w_B t)] + \frac{C_{TxARxA}}{2} [\sin(w_A t + w_B t) - \sin(w_A t - w_B t)] + \frac{G_{TxBRxA}}{2} [1 - \cos(2w_B t)] + \frac{C_{TxBRxA}}{2} [\sin(2w_B t)] \quad (4.9)$$

$$v_{GMULTB}(t) = \frac{G_{TxBRxA}}{2} \quad \text{if } w_{LPF} < \min(2w_B, w_A + w_B, |w_A - w_B|) \quad (4.10)$$

The same conditions apply for the low-pass filter cutoff frequency as before.

To determine the capacitance between transmitter A and receiver A, the received signal is multiplied by a 90° phase shifted version of the reference for transmitter A as follows

$$v_{CMULTA}(t) = G_{TxARxA} \sin(w_A t) \cos(w_A t) + C_{TxARxA} \cos(w_A t) \cos(w_A t) + G_{TxBRxA} \sin(w_B t) \cos(w_A t) + C_{TxBRxA} \cos(w_B t) \cos(w_A t) \quad (4.11)$$

which can be rewritten using trigonometric identities as follows

$$v_{CMULTA}(t) = \frac{G_{TxARxA}}{2} [\sin(2w_A t)] + \frac{C_{TxARxA}}{2} [\cos(2w_A t) + 1] + \frac{G_{TxBRxA}}{2} [\sin(w_B t + w_A t) + \sin(w_B t - w_A t)] + \frac{C_{TxBRxA}}{2} [\cos(w_B t + w_A t) + \cos(w_B t - w_A t)] \quad (4.12)$$

$$v_{CMULTA}(t) = \frac{C_{TxARxA}}{2} \quad \text{if } w_{LPF} < \min(2w_A, w_B + w_A, |w_B - w_A|) \quad (4.13)$$

Consequently, the multiplier output is proportional to the capacitance between receiver A and transmitter A provided the cutoff frequency of the low-pass filter is less than the minimum of $2w_A$, $w_B - w_A$ or $w_B + w_A$.

To determine the capacitance between transmitter B and receiver A, the received signal is multiplied by a 90° phase shifted version of the reference for transmitter B and the multiplier output is

$$v_{CMULTB}(t) = \frac{G_{TxARxA}}{2} [\sin(w_A t + w_B t) + \sin(w_A t - w_B t)] + \frac{C_{TxARxA}}{2} [\cos(w_A t + w_B t) + \cos(w_A t - w_B t)] \\ + \frac{G_{TxBRxA}}{2} [\sin(2w_B t)] + \frac{C_{TxBRxA}}{2} [\cos(2w_B t) + 1] \quad (4.14)$$

$$v_{CMULTB}(t) = \frac{C_{TxBRxA}}{2} \quad \text{if } w_{LPF} < \min(2w_B, w_A + w_B, |w_A - w_B|) \quad (4.15)$$

The above theoretical examination can also be performed for receiver B.

It is important to distinguish FDM impedance tomography from spectroscopic frequency tomography. Dielectric spectroscopy is the technique of obtaining the permittivity of a material at different excitation frequencies [39] and has long been used for material characterisation. As an example, the system developed by Georgakopoulos *et al* is for a frequency range from 10kHz to 1MHz. Various excitation techniques exist including [39]

1. sequentially sweeping the frequency over the desired range
2. white noise generators
3. specific output functions such as the delta function or $\frac{\sin(x)}{x}$

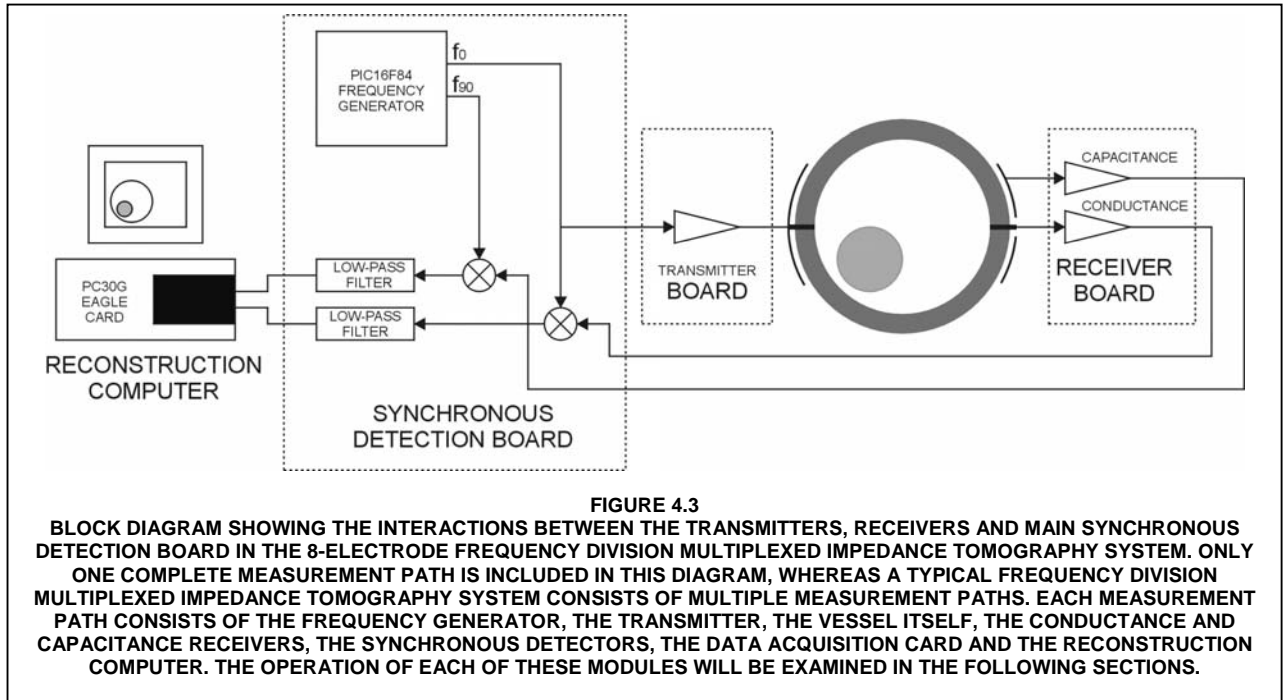
The frequency components are then extracted either in hardware using phase-sensitive demodulation or in the discrete time domain by means of the DTF transform [39].

Indeed, close similarities do exist between the author's research and research conducted at the University of Sheffield [40] where Electrical Impedance Tomographic Spectroscopy (EITS) is used in medical applications to investigate the bio-impedance of certain tissues. Systems developed by the University of Sheffield include a 16-electrode system that uses dedicated current injection and voltage detection electrodes and operates over eight frequencies from 9.6kHz to 1.2MHz at a frame-rate of 67frames/s. Further, a three-dimensional system has been developed consisting of four rings of 16 electrodes to construct a three-dimensional model of the interrogated region [40]. However, the fundamental measurement principle of the author's system is different to that of EITS in that the different frequencies are used as a means of separating the signals from the different transmitters operating in parallel and not as a means of identifying how the complex impedance of a substance changes with frequency.

4.2 Design of an 8-electrode System to Verify the Frequency Division Multiplexed Impedance Tomography Concept

Before work could be started on constructing a laboratory-scale impedance tomography flowmeter, it was necessary to verify whether the FDM concept would actually work. In addition, a comparison to the performance of a TDM impedance tomography system was required in order to assess the loss of resolution resulting from the reduction in available readings. This was achieved through modification of the rig used in the previous project research [1, 2, 3, 4]. The rig is a polyester pipe of inner diameter 220mm and height 300mm. One end of the pipe is sealed so that it retains water when the axis is vertical. The capacitance electrode array is mounted on the external periphery of the vessel and consists of eight galvanised steel plates. At the centre of each plate is a 6mm stainless steel machine screw which is threaded through the pipe wall and ground flush with the inside of the pipe. An earthed outer shield is also provided to prevent external electromagnetic interference.

Figure 4.3 is a block diagram of the major components in a FDM impedance tomography system, although only one complete channel is shown. The following sections will examine the implementation of each of these components in greater detail.



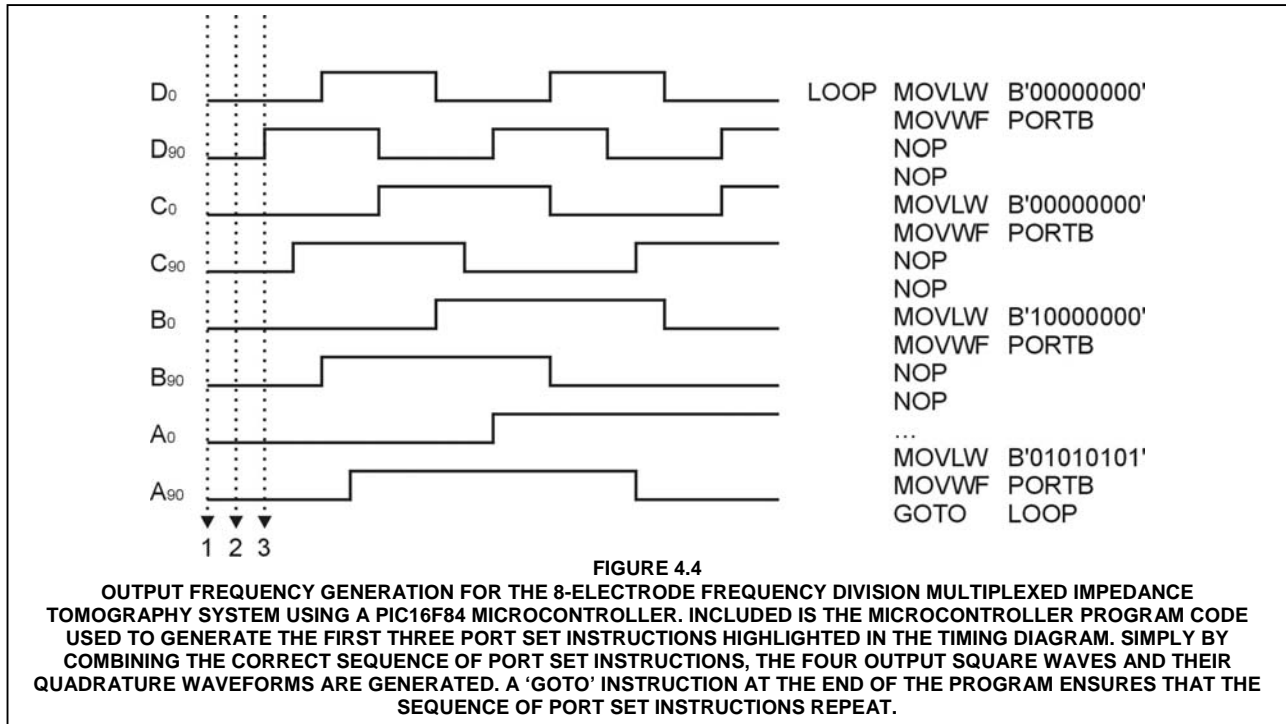
4.2.1 Generation of transmitter frequencies

Although the previous analysis was for sine wave excitation, cost constraints limited the application of the FDM technique to square waves. The use of square wave excitation is still theoretically correct. However, instead of a single frequency component for each transmitter, the square wave outputs consist of a number of additional odd harmonics of the fundamental frequency, as is well known in Fourier analysis [41 pp. 9-45]. The implications of this will be examined at a later stage.

Since the conductance measurements are based on a two-electrode measurement protocol, the contact impedance between the electrode and the substance being imaged becomes an important parameter. As mentioned previously, this contact impedance is the result of electrochemical reactions taking place between the metal conductance probe and the ionic solution. It has been shown that the contact impedance becomes negligible for the case where only one of the phases is conductive, provided a sufficiently high excitation frequency is used [42]. An excitation above 1kHz is required for potable water [42]. In addition, it must not be excessively high so as to avoid unwanted electromagnetic effects and external noise.

The output frequencies are generated using a PIC16F84A microcontroller with a 4MHz crystal. It is necessary to generate four output square waves with 50% duty cycle, as well as four 90° phase shifted versions for the synchronous detection of the capacitance components. A microcontroller provides a convenient and simple solution to the generation of these quadrature waveforms. Port B was used to output the four square waves and their quadrature waveforms. In particular, port B0 was used to output A_0 , port B1 was used to output the quadrature waveform A_{90} , and so on. The full details are provided in the circuit diagram in Appendix B on page 118.

The output waveforms were initially generated by program loops. Tests revealed that this technique was unable to generate frequencies high enough to extract a usefully measurable capacitance component. It was deemed necessary to generate frequencies in the range from approximately 20kHz to 80kHz. In addition, this technique was unable to guarantee the 50% duty cycle and 90° phase shift necessary to accurately separate the capacitance and conductance components. It was therefore decided that the square waves would be generated using a different software approach.



The output square waves are generated by hard-coding the microcontroller port set instructions. Figure 4.4 illustrates this method. Included in figure 4.4 is the microcontroller program code for the first three port set instructions, as well as the last port set instruction in the sequence. The full program code can be found in Appendix E on page 130. Simply by combining the correct sequence of port set instructions, the four output square waves and their quadrature waveforms are generated.

A 'goto' instruction at the end of the sequence ensures that the loop repeats and the square waves are repeated. In order to generate complete waveforms, a sequence of port set instructions corresponding to the lowest common multiple of all the square wave periods must be written. This ensures that when the end of the sequence is reached, all the square waves complete a cycle at the same time and no half cycles exist. The periods of the above square waves are 8, 12, 16 and 20 cycles respectively. The sequence of 240 port set instructions were calculated using a program written in Delphi. They were written to a text file, compiled using MPASM and downloaded using the download software provided with the microcontroller programmer.

One further subtlety of this software is the inclusion of the 'nop' instructions. To ensure that the output waveforms have the correct 50% duty cycle and 90° phase shift, it is necessary to analyse the timing of the instructions. In particular, it was noted that the 'goto' instruction at the end of the sequence requires two instruction cycles to complete. Consequently, if this was not compensated for then the duty cycle and phase shift would be incorrect when the 'goto' instruction was executed. Two 'nop' instructions are included after every port set instruction, except

TABLE 4.1
OUTPUT FREQUENCIES GENERATED WITH A 4MHz
CRYSTAL FOR THE 8-ELECTRODE SYSTEM.

OUTPUT	FREQUENCY
A	25kHz
B	31.25kHz
C	41.67kHz
D	62.5kHz

the last in the sequence, to compensate for this. The last port set instruction is followed by the 'goto' instruction and consequently the correct duty cycle and phase shift is maintained. Using this technique, the output frequencies generated with a 4MHz crystal are given in table 4.1. Further, Appendix G on page 135 lists up to the 9th harmonic of these fundamental frequencies.

4.2.2 Output transmitter details

The load on a transmitter was measured using an RLC meter. In particular, for the rig filled only with seawater, the average resistance of a transmitter-receiver electrode pair is 50Ω. Since each transmitter is required to drive four receiver electrodes in parallel, the approximate load on a transmitter is 12.5Ω. A power op-amp was therefore required. Specifically, the LM675 power op-amp was used for the output transmitter and the circuit details are provided in Appendix B on page 117. Table 4.2 briefly lists the specifications of the LM675 that are appropriate to this application.

TABLE 4.2
LM675 SPECIFICATIONS APPROPRIATE TO THIS APPLICATION.
($V_s = \pm 25V$, $T_A = 25^\circ C$)

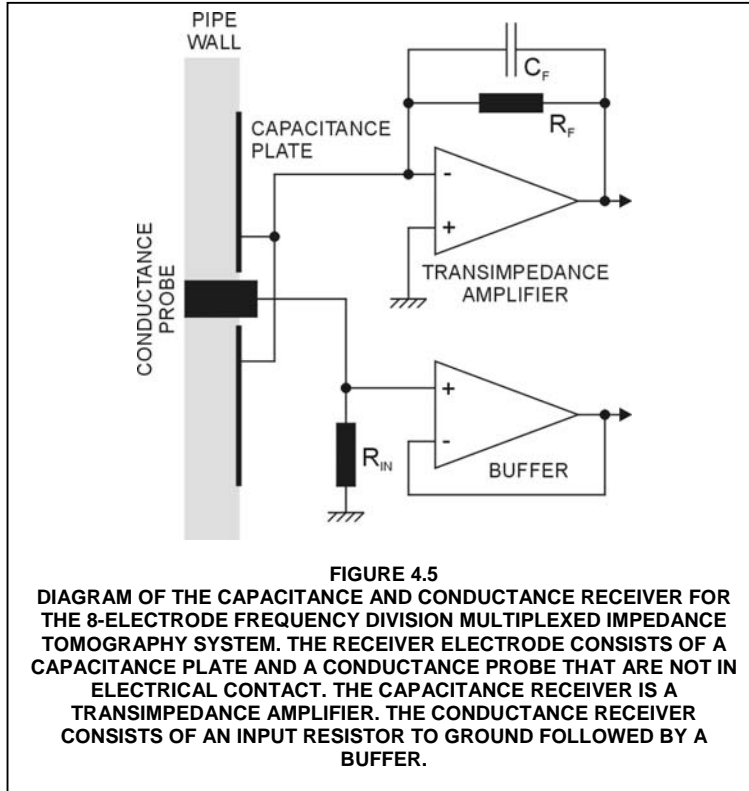
PARAMETER	CONDITION	TYPICAL
OUTPUT POWER	THD = 1%, $f_o = 1kHz$, $R_L = 8\Omega$	25W
OUTPUT VOLTAGE SWING	$R_L = 8\Omega$	$\pm 21V$
GAIN BANDWIDTH PRODUCT	$f_o = 20kHz$, $A_{VCL} = 1000$	5.5MHz
MAXIMUM SLEW RATE		8V/ μs

Although this op-amp was capable of driving the load presented by the rig, the output square waves are distorted as a result of the limited slew rate and the crossover distortion of the op-amp. The effect of this distortion is evident when the output waveforms are monitored using a spectrum analyser. An ideal square wave only has frequency components at the odd harmonics of the fundamental frequency. However, these distortions resulted in significant components at the even harmonics of the fundamental frequency as well. Since the system is attempting to isolate individual frequency components, the generation of additional frequency components is a highly undesirable situation and will be examined in greater detail once the operation of the synchronous detector electronics has been explained.

It was observed that serious corrosion of the conductance probes would take place when only one transmitter was operating. In addition, bubbles would form on the conductance probes. However, the situation improved greatly once all transmitters were operating simultaneously and minimal bubble formation was observed. The consequences of these electrochemical reactions between the seawater and the conductance probes will be examined at a later stage when the drift of the capacitance and conductance readings is analysed.

4.2.3 Conductance and capacitance receiver details

Initially, the conductance probe and the capacitance plate of the receivers were in electrical contact and a single transimpedance amplifier was used to generate an output waveform whose amplitude and phase were a function of the complex impedance of the material within the rig. The individual conductance and capacitance components were then separated at the synchronous detection stage. A power op-amp was required due to the large conductance component. Specifically, a LM6313 high speed, high power op-amp was used to perform this task. The peak output current of the LM6313 is $\pm 300\text{mA}$. However, initial tests revealed that although good results were achieved for the measurement of the conductance component, the changes in capacitance were less detectable and were sensitive to conductance variations. Hence it was decided that the conductance and capacitance receiver paths should be isolated. A larger hole was drilled in the centre of the capacitance plates to ensure that there was no electrical contact between the conductance probes and the capacitance plates.



The conductance receiver consists of an input resistor to ground followed by a buffer. As the conductance of the material within the rig changes, so the potential across the input resistor R_{IN} changes. In particular, as the conductance increases, so the amplitude of the output waveform increases. The input resistor was chosen to ensure that the buffer output did not saturate when all the transmitters were operating simultaneously. Figure 4.5 is a simplified diagram of the capacitance and conductance receiver.

The capacitance receiver is a transimpedance amplifier. In the ideal case of capacitor input and feedback elements, the output voltage V_O is given by the following equation relating the feedback capacitance C_F to the unknown capacitance C_X for a transmitter voltage V_E [6]

$$V_O = -\frac{C_X}{C_F} V_E \quad (4.16)$$

Ideally, C_F should be chosen as small as possible. For capacitance feedback to dominate, the feedback resistor and capacitor component values must be chosen so that $\frac{1}{\omega C_F} \ll R_F$ [16]. The resistive feedback is required to ensure that the op-amp output does not saturate as a result of the DC bias current charging the feedback capacitor [6].

Unlike standard tomography systems, the receiver outputs for FDM impedance tomography consist of a superposition of waveforms corresponding to the signals from the different transmitters. In addition, square waves are used instead of sine waves. Although it may have been possible to calculate theoretically ideal resistor and capacitor values for the receiver, the application of these values to this specific application would not have been practical. Hence, the feedback capacitor and resistor values were chosen as follows: with only the lowest frequency transmitter operating, the resistor and capacitor value were chosen so that the output from the square wave transition decayed to zero at roughly the same time as the next square wave transition. This ensured that a satisfactory signal level was received for all transmitters. Then with all transmitters operating simultaneously, this RC combination remained fixed while the feedback resistor value was reduced to ensure that the output of the amplifier did not saturate. The circuit details of the conductance and capacitance receiver are given in Appendix B on page 117.

4.2.4 Synchronous detection of receiver signals

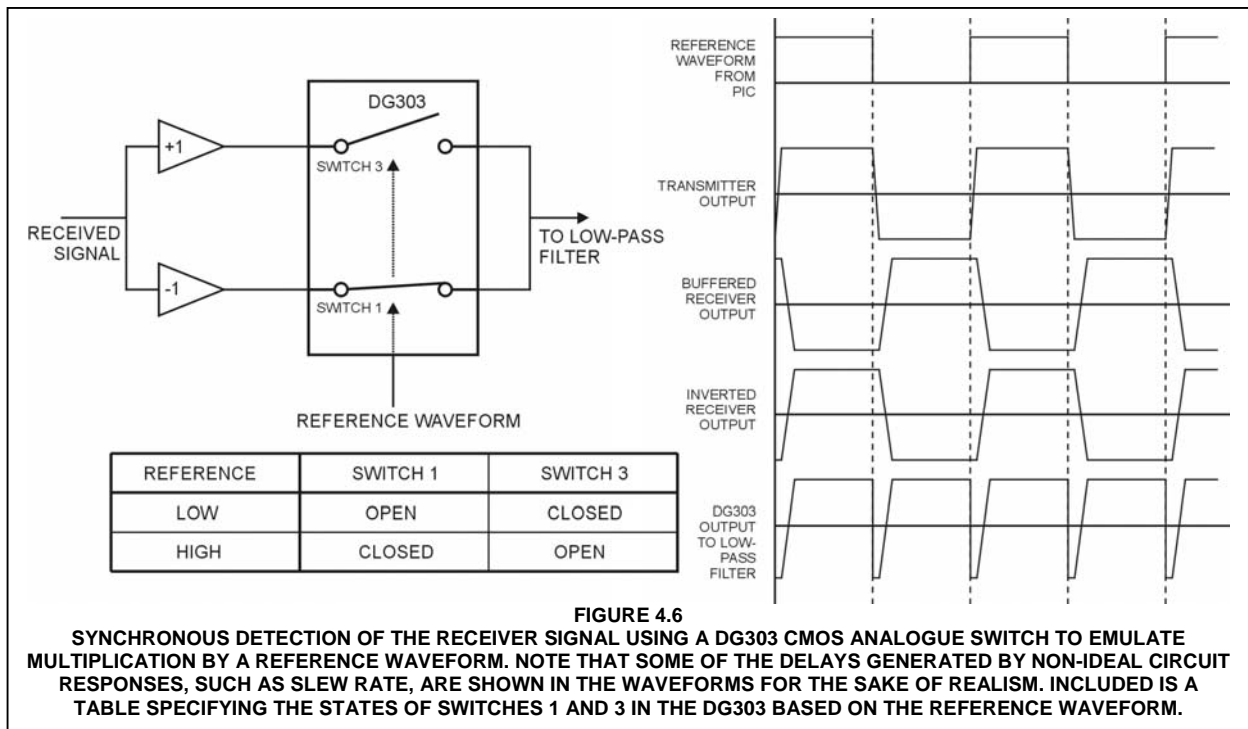
Phase-sensitive detection can be achieved using either multiplication or switching techniques. Results have shown that the multiplying phase-sensitive detector produces more accurate results since the switching technique suffers from the following problems [43]:

1. the charge injection from the CMOS switches
2. the introduction of odd harmonics

In addition, it has been shown that the multiplication demodulator is 1.2 times faster than the switch-based phase-sensitive demodulator for the same level of accuracy [43]. However, since the FDM impedance tomography system uses large numbers of multipliers in parallel, cost constraints limited the electronics to CMOS switch-based techniques. In particular, a DG303 dual single-pole-double-throw CMOS analogue switch was used.

The following explanation of switch-based synchronous detection will consider the received signals from receiver A with only transmitter A operating. All appropriate waveforms are included in figure 4.6 on page 29, which shows the specific results for the received conductance signal. Firstly, the received signal is buffered and inverted to produce the original waveform and an inverted version with a 180° phase shift between the two. These signals are connected to the inputs of two switches whose outputs are connected together. The states of these two switches are opposite and are driven by A_0 from the PIC16F84, which is also used to drive the transmitter. The operation of these switches is given in the table within the figure. For example, when A_0 is high, switch 1 closes and switch 3 opens and the inverted signal is connected to the low-pass filter. Then when A_0 is low, switch 3 closes and switch 1 opens and the buffered signal is connected to the low-pass filter. If there was no phase shift in the system and the transmitter outputs were perfect square waves, then the output of the DG303 would be a pure DC voltage proportional to the conductance between transmitter A and receiver A. However, due to the phase shifts introduced by the different stages in the system and the limited slew rate of the output transmitters, the DG303 output actually consists of a DC voltage with an AC component, as can be seen in figure 4.6. It is the task of the low-pass filter to extract the DC component from this signal. It is evident in figure 4.6 that an inverted DG303 output is equivalent to the multiplication of the receiver signal by the reference waveform.

The situation for capacitance detection is very similar to the above except that the switches are controlled by A_{90} instead of A_0 , so as to produce an output proportional to the capacitance component of the received signal. Since the DG303 is a dual single-pole-double-throw CMOS analogue switch, the capacitance demodulation for a particular transmitter-receiver electrode pair is performed using the same integrated circuit as the conductance demodulation detailed above.



When all transmitters are operating simultaneously, the received signal consists of a superposition of components from the different transmitters. As before, the received signal is buffered and inverted. These signals are then applied to four DG303 CMOS analogue switches in parallel. The first DG303 has A_0 and A_{90} as control inputs to produce voltages proportional to the conductance and capacitance between transmitter A and receiver A respectively. The second DG303 has B_0 and B_{90} as control inputs to produce voltages proportional to the conductance and capacitance between transmitter B and receiver A respectively. The third DG303 has C_0 and C_{90} as control inputs to produce voltages proportional to the conductance and capacitance between transmitter C and receiver A respectively and finally, the fourth DG303 has D_0 and D_{90} as control inputs to produce output voltages proportional to the conductance and capacitance between transmitter D and receiver A respectively. All this circuitry is duplicated for receiver B, C and D so as to produce voltages proportional to all the different transmitter-receiver electrode pairs in parallel.

Selection of the low-pass filter cutoff frequency is an important design parameter. If it is too low, then the settling time will be too long thus limiting the effective frame-rate of the system. Since a frame-rate of 200frames/s is required, the cutoff frequency of the low-pass filter must be greater than 200Hz. If it is too high, then the passband of the low-pass filter may overlap with the undesired harmonics in the DG303 output resulting in a poor signal-to-noise ratio [44]. Since the minimum frequency separation of the four transmitter frequencies used is 6.25kHz, the cutoff of the low-pass filter must be less than 6.25kHz. Consequently, an initial cutoff frequency of 625Hz was chosen. However, only first-order low-pass filters were implemented and therefore a fairly substantial 6.25kHz ripple was still evident on the DC output of the low-pass filters. To reduce this ripple to a level where the accuracy of the reconstruction was not affected, it was necessary to set the cutoff frequency of the low-pass filters to 3Hz, thus limiting the frame-rate of this prototype system to 3frames/s. Clearly, this limitation would need to be addressed in the final system in order to take full advantage of the FDM technique, but it was deemed satisfactory in the short term since it was only necessary to prove that the concept actually worked. The circuit details of the synchronous detection circuitry are given in Appendix B on page 119.

4.2.5 Capture of capacitance and conductance readings and the reconstruction results achieved

The voltages are captured using an Eagle PC30G data acquisition card plugged into the reconstruction computer, as illustrated in figure 4.3 on page 24. This card supports 16 analogue input voltages in the range from -10V to 10V and can achieve a sampling rate of 100ksamples/s , provided the data is streamed directly to memory under the supervision of the Direct Memory Access (DMA) controller. It also provides three 8-bit bi-directional digital input-output ports. The voltages are sampled using single ended input sampling. Although this causes the sampled readings to be more sensitive to noise, the lead lengths connecting the PC30G to the multiplexing board were kept short so as to minimise this effect. In addition, analogue ground was isolated from digital ground plane noise through an inductor.

Two DG506A 16-input CMOS analogue multiplexers were used to multiplex the 16 capacitance and 16 conductance voltages. The address lines of the multiplexers were driven directly by digital output port A of the PC30G card. In addition, the enable lines of the analogue multiplexers were driven by a digital output of the card. The two analogue outputs of the multiplexers were connected to analogue input channel 0 and 1 respectively. A complete frame of voltages was captured as follows:

1. Port A was set under software control so as to access address 0 on the multiplexers and the multiplexers were enabled.
2. After a delay, the voltages on channel 0 and channel 1 were sampled under software control and stored in the corresponding locations of the complete frame.
3. Port A was then incremented to access the next address and the process was repeated until a complete frame of capacitance and conductance readings had been captured.

The circuit diagram of the multiplexing card can be found in Appendix B on page 121.

A minimum bubble diameter corresponding to 20% of the pipeline diameter was used in the author's previous research, since capacitance tomography systems typically produce low-resolution images and the readings were subject to measurement noise and drift. To ensure a consistent comparison, the training database for the FDM impedance tomography system was generated using the same bubble positions and minimum bubble sizes, namely 20%, 40% and 50% of the pipeline diameter respectively, for both air and gravel phases. In addition, the software developed previously was used to generate training and test databases and train neural networks to perform the reconstruction for the FDM impedance tomography system. The only modifications included changing the data capture software to capture readings from the PC30G card instead of the serial port, and modifying the neural networks to have 32 inputs instead of the previous 56. Since the program code is effectively the same as that developed previously, it will not be examined here and has not been included in the appendices. Instead, the neural network reconstruction results will be examined, and a comparison to the previous results achieved using a TDM impedance tomography system will be provided.

It is important to note that a consistent comparison was performed between the TDM impedance tomography system and the FDM impedance tomography system, since the same pipeline section and electrode array were used for both sets of tests. Further, both systems were tested with the same air-gravel-seawater phase configurations and the same neural network topologies were used to perform the reconstructions of both systems. Hence, any difference in the reconstruction accuracy is related specifically to the measurement technique employed, namely either TDM or FDM impedance tomography.

Firstly, to verify the operation of the system, a single-layer feed-forward neural network was trained using gradient descent to perform a two-phase air-seawater image reconstruction. The operation of this neural network was examined in figure 3.1 on page 12. Early stopping was used to prevent over-fitting. Since the neural network performance depends largely on the random initialisation of the network weights, three networks were trained and their results averaged to give a more representative indication of system performance. The neural network reconstruction results are detailed in table 4.3 below, which also gives the corresponding performance results of the TDM impedance tomography system. The white cells indicate which of the compared cell entries have the better performance.

TABLE 4.3
COMPARISON BETWEEN THE PERFORMANCE OF THE FREQUENCY DIVISION MULTIPLEXED IMPEDANCE TOMOGRAPHY SYSTEM AND A STANDARD TIME DIVISION MULTIPLEXED IMPEDANCE TOMOGRAPHY SYSTEM FOR A TWO-PHASE AIR-SEAWATER IMAGE RECONSTRUCTION USING A SINGLE-LAYER FEED-FORWARD NEURAL NETWORK. ALL RESULTS REPRESENT THE ABSOLUTE ERROR AND ARE ROUNDED OFF TO TWO SIGNIFICANT DIGITS.

PERFORMANCE MEASURE	TIME DIVISION MULTIPLEXED IMPEDANCE TOMOGRAPHY SYSTEM	FREQUENCY DIVISION MULTIPLEXED IMPEDANCE TOMOGRAPHY SYSTEM
THRESHOLD ERROR (%)	14	9.1
AIR VOLUME FRACTION ERROR (%)	7.9	5.0
WATER VOLUME FRACTION ERROR (%)	7.9	5.0

In a surprising result, the FDM impedance tomography system outperforms the TDM impedance tomography system on all the performance criteria considered. Since only 32 readings are available for the new system instead of 56, it was expected that the resolution, and consequently the accuracy, of the reconstructed images would have deteriorated.

One possible explanation for this improvement in performance is that the FDM impedance tomography system is using separate capacitance and conductance receivers. Previously, a single receiver was used for both the capacitance and conductance signals. Since there is a large conductance component from the seawater, the gain on this receiver was fairly low to ensure the op-amp output did not saturate. Consequently, the received capacitance component was very small and was affected by the conductance signal. By using separate receivers, each receiver can be tuned to give optimal signal detection for that particular measurement, hence a much larger capacitance gain was achieved. In addition, the interference between the two measurements was reduced. These results were confirmed when a single-layer feed-forward neural network was trained using gradient descent to perform a two-phase gravel-seawater image reconstruction, the results of which are given in table 4.4 below.

TABLE 4.4
COMPARISON BETWEEN THE PERFORMANCE OF THE FREQUENCY DIVISION MULTIPLEXED IMPEDANCE TOMOGRAPHY SYSTEM AND A STANDARD TIME DIVISION MULTIPLEXED IMPEDANCE TOMOGRAPHY SYSTEM FOR A TWO-PHASE GRAVEL-SEAWATER IMAGE RECONSTRUCTION USING A SINGLE-LAYER FEED-FORWARD NEURAL NETWORK. ALL RESULTS REPRESENT THE ABSOLUTE ERROR AND ARE ROUNDED OFF TO TWO SIGNIFICANT DIGITS.

PERFORMANCE MEASURE	TIME DIVISION MULTIPLEXED IMPEDANCE TOMOGRAPHY SYSTEM	FREQUENCY DIVISION MULTIPLEXED IMPEDANCE TOMOGRAPHY SYSTEM
THRESHOLD ERROR (%)	13	8.7
GRAVEL VOLUME FRACTION ERROR (%)	4.5	4.4
WATER VOLUME FRACTION ERROR (%)	4.5	4.4

A single-layer feed-forward neural network with a 1-of-C output encoding was trained using Resilient back-propagation to perform a three-phase air-gravel-seawater image reconstruction. The operation of this neural network was examined in figure 3.2 on page 13. The decision to use Resilient back-propagation was based on the results of the previous research. Once again, three networks were trained and the average results are given in table 4.5 below.

TABLE 4.5
COMPARISON BETWEEN THE PERFORMANCE OF THE FREQUENCY DIVISION MULTIPLEXED IMPEDANCE TOMOGRAPHY SYSTEM AND A STANDARD TIME DIVISION MULTIPLEXED IMPEDANCE TOMOGRAPHY SYSTEM FOR A THREE-PHASE AIR- GRAVEL-SEAWATER IMAGE RECONSTRUCTION USING A SINGLE-LAYER FEED-FORWARD NEURAL NETWORK WITH A 1-OF-C OUTPUT ENCODING. ALL RESULTS REPRESENT THE ABSOLUTE ERROR AND ARE ROUNDED OFF TO TWO SIGNIFICANT DIGITS.

PERFORMANCE MEASURE	TIME DIVISION MULTIPLEXED IMPEDANCE TOMOGRAPHY SYSTEM	FREQUENCY DIVISION MULTIPLEXED IMPEDANCE TOMOGRAPHY SYSTEM
THRESHOLD ERROR (%)	20	13
AIR VOLUME FRACTION ERROR (%)	9.1	2.9
GRAVEL VOLUME FRACTION ERROR (%)	5.6	5.3
WATER VOLUME FRACTION ERROR (%)	12	5.6
SUM VOLUME FRACTION ERROR	27	14

An additional performance assessment criterion is included labelled 'sum volume fraction error'. This is simply the sum of the seawater, gravel and air volume fraction errors. Since the intended application of the system is the accurate measurement of the volume fractions, this measure provides a simple means of comparing the volume fraction accuracies of two networks without having to compare the individual volume fraction error percentages for each of the three phases. As can be seen from table 4.5, the FDM impedance tomography system is capable of distinguishing between the gravel and air phases and actually produces more accurate results than the TDM impedance tomography system examined previously. In particular, the accuracy of the seawater and air volume fraction predictions has improved considerably. It is believed that this is a result of isolating the conductance and capacitance measurements on completely separate measurement channels.

It has been shown that improved volume fraction predictions can be obtained by training a neural network to predict the volume fractions of the different phases directly without having to first perform an image reconstruction [3, 4]. A double-layer feed-forward neural network was trained using gradient descent to predict the volume fractions of a three-phase air-gravel-seawater mixture directly. The operation of this neural network was examined in figure 3.3 on page 14. The number of hidden layer neurons was chosen to be 25 based on the results achieved for the TDM impedance tomography system. Early stopping was employed to prevent over-fitting and table 4.6 provides a comparison between the performances of the FDM impedance tomography system and the TDM impedance tomography system. As is evident from table 4.6, the performance of the volume fraction predictor is also better for the FDM impedance tomography system.

TABLE 4.6
COMPARISON BETWEEN THE PERFORMANCE OF THE FREQUENCY DIVISION MULTIPLEXED IMPEDANCE TOMOGRAPHY SYSTEM AND A STANDARD TIME DIVISION MULTIPLEXED IMPEDANCE TOMOGRAPHY SYSTEM FOR A THREE-PHASE AIR- GRAVEL-SEAWATER VOLUME FRACTION PREDICTION USING A DOUBLE-LAYER FEED-FORWARD NEURAL NETWORK. ALL RESULTS REPRESENT THE ABSOLUTE ERROR AND ARE ROUNDED OFF TO TWO SIGNIFICANT DIGITS.

PERFORMANCE MEASURE	TIME DIVISION MULTIPLEXED IMPEDANCE TOMOGRAPHY SYSTEM	FREQUENCY DIVISION MULTIPLEXED IMPEDANCE TOMOGRAPHY SYSTEM
AIR VOLUME FRACTION ERROR (%)	8.9	7.4
GRAVEL VOLUME FRACTION ERROR (%)	5.0	4.7
WATER VOLUME FRACTION ERROR (%)	6.5	4.8
SUM VOLUME FRACTION ERROR	20	17

4.2.6 Limitations of the current system

Although the above results for the FDM impedance tomography system are promising, they do illustrate certain limitations of the prototype system. Firstly, having to set the cutoff frequency of the low-pass filter to only 3Hz limits the frame-rate of the system to only 3frames/s and consequently negates any possible speed advantages achieved through the use of parallelism. Clearly, a low-pass filter with a steeper rolloff would be required in the next system. Alternate low-pass filtering techniques were tested on the prototype rig before being implemented on the laboratory-scale rig. Specifically, the LMF100 dual second-order switched capacitor filter was tested.

Secondly, the transmitter outputs contain additional frequency harmonics due to the distortions introduced by the power op-amp. These additional frequency harmonics result in the generation of unwanted frequency components on the DG303 outputs and impose greater limitations on the low-pass filtering stage to ensure minimal ripple on the output voltages. Research was done at the Cape Technikon to assess the performances of different power op-amps [45]. Since only a square wave is required, various MOSFET drivers were also tested. This research concluded that MOSFET drivers should be used in the laboratory-scale rig. The operation of these drivers will be examined at a later stage.

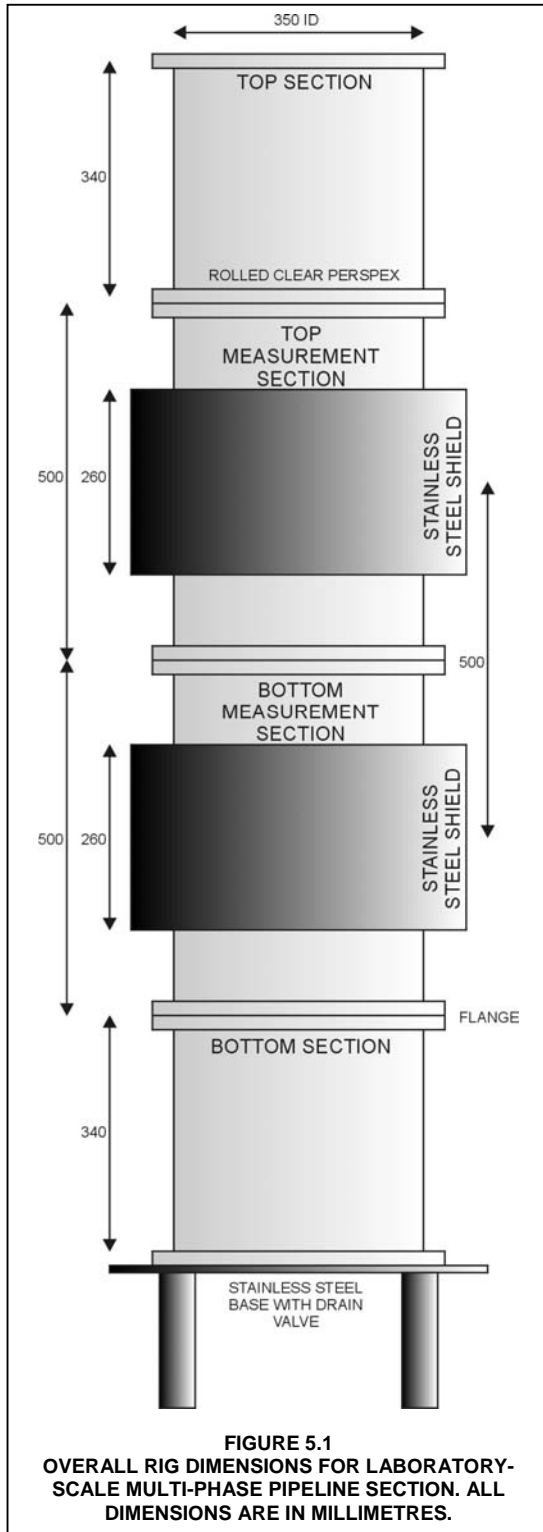
Finally, the sampling speed of the data acquisition card would need to be improved. According to the PC30G manual, the maximum sampling rate that can be achieved using single voltage captures under software control is typically 3ksamples/s. Although the card is rated at 100ksamples/s, this is only achieved through the use of Direct Memory Access (DMA). Consequently, the data acquisition software for the laboratory-scale rig would need to make use of DMA in order to achieve the high frame-rate required for the intended application.

CHAPTER 5

FREQUENCY DIVISION MULTIPLEXED IMPEDANCE TOMOGRAPHY OF A 16-ELECTRODE SYSTEM

The following chapter details the development of a dual-plane 16-electrode FDM impedance tomography system.

5.1 Construction of a Laboratory-Scale Rig to Simulate a Pipeline Section



A laboratory-scale rig was constructed to assess the performance of the FDM impedance tomography system on simulated flows. The following sections examine the selection of certain parameters during the design of the rig.

5.1.1 Reasoning behind certain rig dimensions

The rig diameter was chosen to be equal to the pipeline diameter of the industrial airlift section where the impedance tomography system is to be installed. The major rig dimensions are provided in figure 5.1. The rig consists of four distinct sections, namely the two measurement planes, and a top and a bottom as can be seen in the accompanying figure. The top and bottom sections are provided for the flow simulation tests. Specifically, they provide a region before and after the measurement planes for the bubbles to accelerate and decelerate. Ideally, the bubble should be travelling at constant velocity while moving through the measurement sections since cross-correlation measures the average velocity between the two measurement planes. Included in the bottom section are two alignment strips used for the exact positioning of test cases. The application of these alignment strips will be illustrated at a later stage.

An advantage of constructing different sections is the flexibility of the design. As an example, the top section can be placed between the two measurement sections to increase the measurement plane separation. Also, each measurement plane can be tuned and tested separately simply by constructing a blanking-off disk. The measurement section is then simply a larger version of the prototype rig thus permitting a consistent comparison between the performance of the 8-electrode prototype and the 16-electrode laboratory-scale rig. The sections are joined using stainless steel flanges and are sealed with 3mm rubber insertion gaskets. The entire rig is supported on a raised stainless steel base and a drain valve is provided for cleaning purposes. All metal fittings were



FIGURE 5.2
PHOTOGRAPH OF LABORATORY-SCALE MULTI-PHASE
PIPELINE SECTION.

constructed from stainless steel due to its corrosion resistance. A photograph of the assembled laboratory scale rig can be seen in figure 5.2.

Since the electrodes of a capacitance tomography system are typically mounted outside the vessel, the measured inter-electrode capacitances vary non-linearly with respect to the permittivity of the enclosed material because of the capacitance of the insulating pipe wall [46]. Ideally, the permittivity of the pipe wall material should be small to minimise the field divergence of the capacitance tomography system [47, 48]. Clear acrylic was used for the construction of the vessel. This was necessary to enable visual verification of the reconstruction results. Further, it is essential that training and test databases can be generated quickly and easily since neural networks are used to perform the reconstruction. By using clear acrylic for the pipeline, the task of aligning a specific test or training configuration was simplified.

The selection of the pipe wall thickness is an important design parameter. If it is too thick, then the

capacitance measurements become highly non-linear and the sensing fields of adjacent electrodes are almost completely confined to the pipe wall [47]. In addition, the capacitance-sensing field is more sensitive for a thinner pipe wall [48]. Ultimately, the pipe wall material and thickness is determined by the practical working conditions, such as pressure, corrosion, abrasion, temperature and so on [48]. In this application, the pipe wall thickness was set by the strength required to support the large volume of water contained within the rig. Specifically, the rig will hold approximately 160 litres of seawater when full. Consequently the rig was constructed using 19mm rolled acrylic.

Cross-correlation flow measurement depends on the successful detection of disturbances at both the top and bottom measurement planes. In practice, these disturbances will gradually change on moving downstream. If the position of the downstream sensor were reasonably close to that of the upstream sensor then the patterns or signals generated would be sufficiently similar to be recognised by the cross-correlator [12]. Generally, the selection of the plane separation is a compromise between the similarity of the flow patterns and the resolution with which the transit time τ can be determined [12].

Interference between the electric fields of the two systems requires a large plane separation [43]. Various simulation studies have produced varying results regarding the desired plane separation for dual-plane cross-correlation flowmeters. Loh *et al* show that a separation of at least one pipe diameter is required to reduce the cross-talk to approximately 1% for a resistance tomography system [49]. In contrast, Beck and Plaskowski state that a separation of three to four pipe diameters is necessary [50]. Ultimately, it was the physical size and cost that

limited the laboratory-scale rig to a separation of 500mm, which only represents 1.4 pipe diameters separation. This will be examined in greater detail when the problem of cross-plane interference is addressed at a later stage.

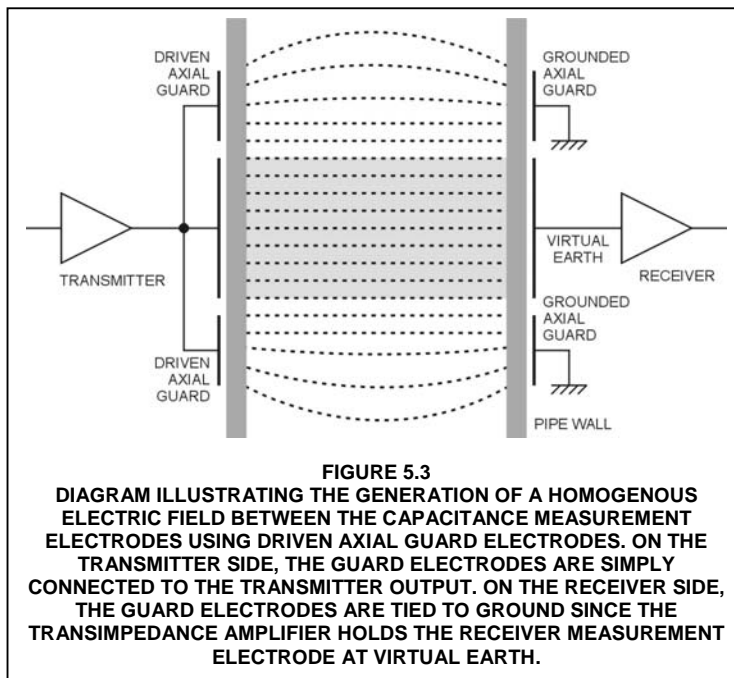
5.1.2 Design of transmitter and receiver electrodes and guarding

The range and accuracy of the capacitance measurement circuitry determines the number of electrodes used. Standing capacitances refer to the capacitances between electrodes when the vessel is filled only with the low permittivity material. Clearly, the standing capacitances of the diametrically opposite electrodes represent the smallest capacitance that must be reliably measured by the electronics. Only by increasing the electrode length, can more electrodes be incorporated and the image resolution increased [11]. Typically, capacitance tomography systems use 12 electrodes giving 66 independent measurements. Since the FDM protocol decreases the number of measurements available for a given number of electrodes, the number of capacitance plates was increased to 16. Consequently, 64 independent capacitance measurements are obtained for a 16-electrode FDM tomography system.

Three types of electrode configurations are typically used for capacitance tomography systems, namely [51]

1. grounded rings above and below the central measurement electrodes
2. driven segmented axial guards above and below the central measurement electrodes, where the guard electrodes are driven to the same potential as the corresponding measurement electrode
3. a single ring of measurement electrodes without guarding

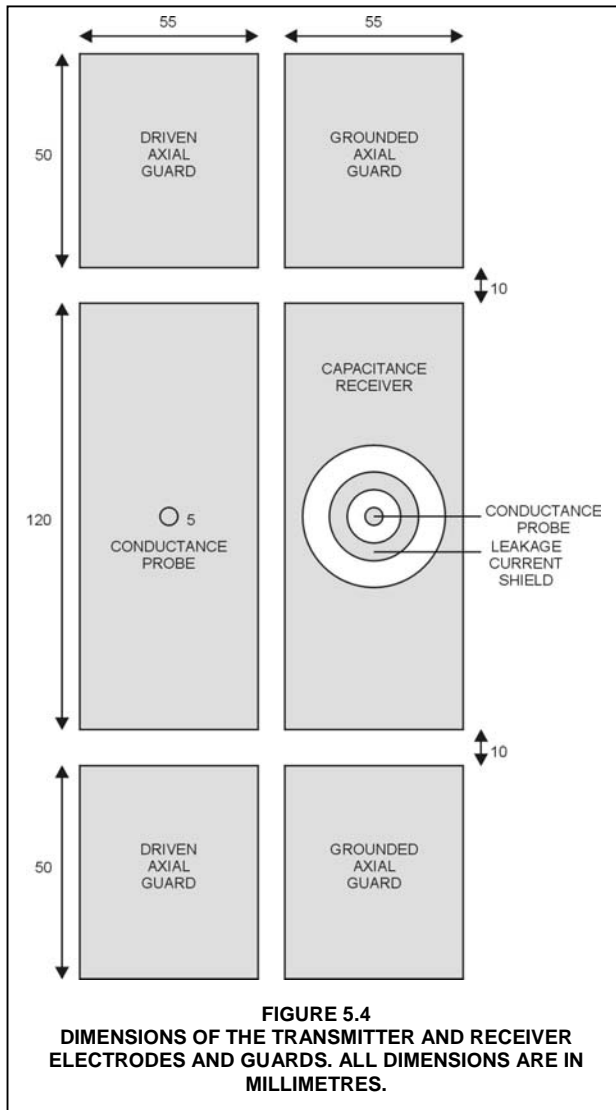
Since the electric field of a capacitance tomography sensor is three-dimensional and the reconstructed images represent a two-dimensional approximation of that field, it is important to ensure the generation of a homogenous electric field. In addition, the measurement space must be focussed. The particular guard electrode design employed has a significant influence on these specifications [51]. Consequently, numerous simulations have been performed analysing the merits of each of these different approaches. These simulations typically use finite element techniques to model the three-dimensional electric field of the above three sensor configurations. However, contrasting results have been achieved and so the following section will briefly compare the major results of a few of these simulations.



Although there is general agreement that the use of driven axial guard electrodes in both capacitance and resistance tomography improve the axial evenness of the electric field or current distribution, different opinions exist regarding whether driven axial guards increase or decrease the measurement volume. Figure 5.3 illustrates the generation of a homogenous electric field for capacitance tomography using driven axial guard electrodes. As is evident in figure 5.3, the electric field generated between the measurement electrodes approximates a two-dimensional field, thus improving the validity of the reconstruction algorithm.

In addition, it has been shown that driven axial guards improve the axial resolution and measurement sensitivity [11]. Further, driven axial guards are normally essential for large diameter pipelines to ensure that the standing capacitances between diametrically opposite electrodes can be measured [11].

In terms of the measurement volume, Yan *et al* [51] and Xu *et al* [52] conclude that the measurement volume produced by driven axial guards is the largest of the three possible electrode configurations considered. Wang [53] confirmed this result for the case of resistance tomography. In contrast, Ma *et al* [54, 55] illustrate the compression of the measurement volume through the use of driven axial guards for resistance tomography. For the laboratory-scale rig being developed, guard electrodes were provided for the capacitance plates but no guarding was provided for the conductance probes. This ensured that reasonable standing capacitances were measured and that a near-parallel electric field improved the accuracy of the reconstructed images.



The axial length of the measurement volume plays an important role in the cross-correlation algorithm. In particular, it affects the spatial filtering and hence the bandwidth of the signals to be correlated [50]. The effective cutoff frequency of capacitance electrodes of length 'a' at a flow velocity 'u' is [50]

$$f_c = \frac{u}{a} \quad (5.1)$$

Small capacitance electrodes are desirable in order to obtain wide spatial bandwidth signals [56] as particles pass into and out the sensing volume [57]. However, short electrodes result in small capacitance readings and the measurement volume must be large relative to the size of the component particles to ensure an accurate volume fraction measurement [57]. Hence, a compromise is made. The dimensions of the transmitter and receiver electrode are detailed in figure 5.4.

A capacitance plate length of 120mm was chosen. When this electrode array was designed, the intended application was for flow velocities up to 12m/s. This has since been increased to 20m/s but the choice of the electrode length will be explained in terms of the initial specification. An object moving at 12m/s will pass through each measurement section in a time of 10ms. Provided the impedance tomography system performs on-line reconstructions at a speed greater than

100frames/s, this object is guaranteed to be captured at least once. In terms of the spatial filtering effect of the electrode, it was assumed that the measurement volume is confined to the measurement plate, namely 120mm. For a maximum velocity of 12m/s, the highest frequency components generated will be 100Hz, as calculated using equation 5.1 above. To satisfy the Nyquist sampling theorem, the impedance tomography system is required to capture data and perform reconstructions at 200frames/s.

Hence, the minimum required frame-rate is 200frames/s for this particular application. The widths of the plates were fixed by the circumference of the pipeline for a given number of electrodes.

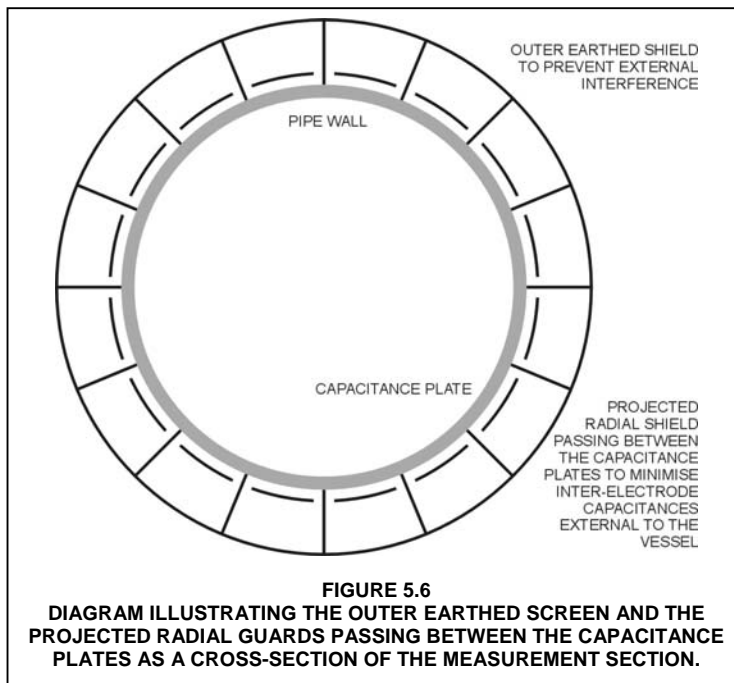


FIGURE 5.5
PHOTOGRAPH OF THE PIPELINE FROM ABOVE WITH THE 16 STAINLESS
STEEL CONDUCTANCE PROBES VISIBLE AT THE TOP AND BOTTOM
MEASUREMENT SECTIONS.

Stainless steel 5mm machine screws were used for the conductance probes to ensure a high electrical conductivity in comparison to the seawater [17]. In addition, stainless steel has the advantages of low cost, ease of fabrication and resistance to corrosion and abrasion [42]. Although standard resistance tomography systems typically use plate electrodes whose dimensions are chosen to optimise the compromise between current injection and voltage detection [58], results can be achieved for point electrodes. In general, the current injection electrodes of a resistance tomography system should be as large as possible to ensure an even current distribution within the vessel [27]. In contrast, the

voltage measurement electrodes should be as small as possible to avoid averaging over several equipotential values [27]. A cause for concern was the fact that a potential difference will exist between the conductance probe and the capacitance plate. The voltage on the conductance probe will vary with the received signal, whereas the capacitance plate will be held at virtual earth by the transimpedance amplifier. Hence it was necessary to keep the conductance probe as small as possible so as to minimise any interference between the capacitance plates and the conductance probes since both measurements are performed simultaneously. Although the conductance probes are invasive, they are designed to be non-intrusive so as not to influence the flow parameters [59]. Figure 5.5 is a photograph of the conductance probes from above.

As can be seen in figure 5.4 on page 37, the conductance probes for the transmitters are in electrical contact with the capacitance plates. For the receiver, a grounded metal ring surrounds the conductance probe. In the previous system there existed a leakage current between the conductance probe and the capacitance plate. Specifically, if there were any moisture in the region of the conductance probe then a leakage current would flow from the conductance probe to the capacitance plate. A potential difference exists between the conductance probe and the capacitance plate and therefore a current will flow. Since the current through the capacitance is very small, any leakage current seriously distorts the results. By placing a grounded metal ring around the conductance probe, any leakage currents will flow to ground and the problem should be solved.



An earthed screen mounted on the outside of the measurement electrodes is used to reject external noise [51]. Radial shields isolate the measurement electrodes to ensure that the inter-electrode capacitances external to the vessel are minimal [46]. They also have the advantage of reducing the standing capacitances of the adjacent electrode pairs thus reducing the required dynamic range of the capacitance sensing circuitry [47, 56]. By analysing the measurement combinations of a FDM impedance tomography protocol, it can be seen that a significant proportion of the measurements are for adjacent transmitter-receiver electrode pairs. The radial shields ensure that these capacitance measurements do take place through the

vessel. Figure 5.6 illustrates the outer earthed screen, as well as the projected radial shields passing between the capacitance plates. Appendix D on page 129 contains tables of the capacitance and conductance readings of the different electrode combinations in the laboratory-scale rig.

5.2 Electronic Design of a 16-electrode Frequency Division Multiplexed Impedance Tomography System

Due to the high level of parallelism in the FDM impedance tomography technique, it was decided that the 16-electrode system should be broken down into separate fundamental modules where each module is then replicated according to the number of electrodes in the system. Figure 5.7 on page 40 is a block diagram illustrating the interaction of these individual modules. The numbers next to each connection specify the number of signal lines between the corresponding modules.

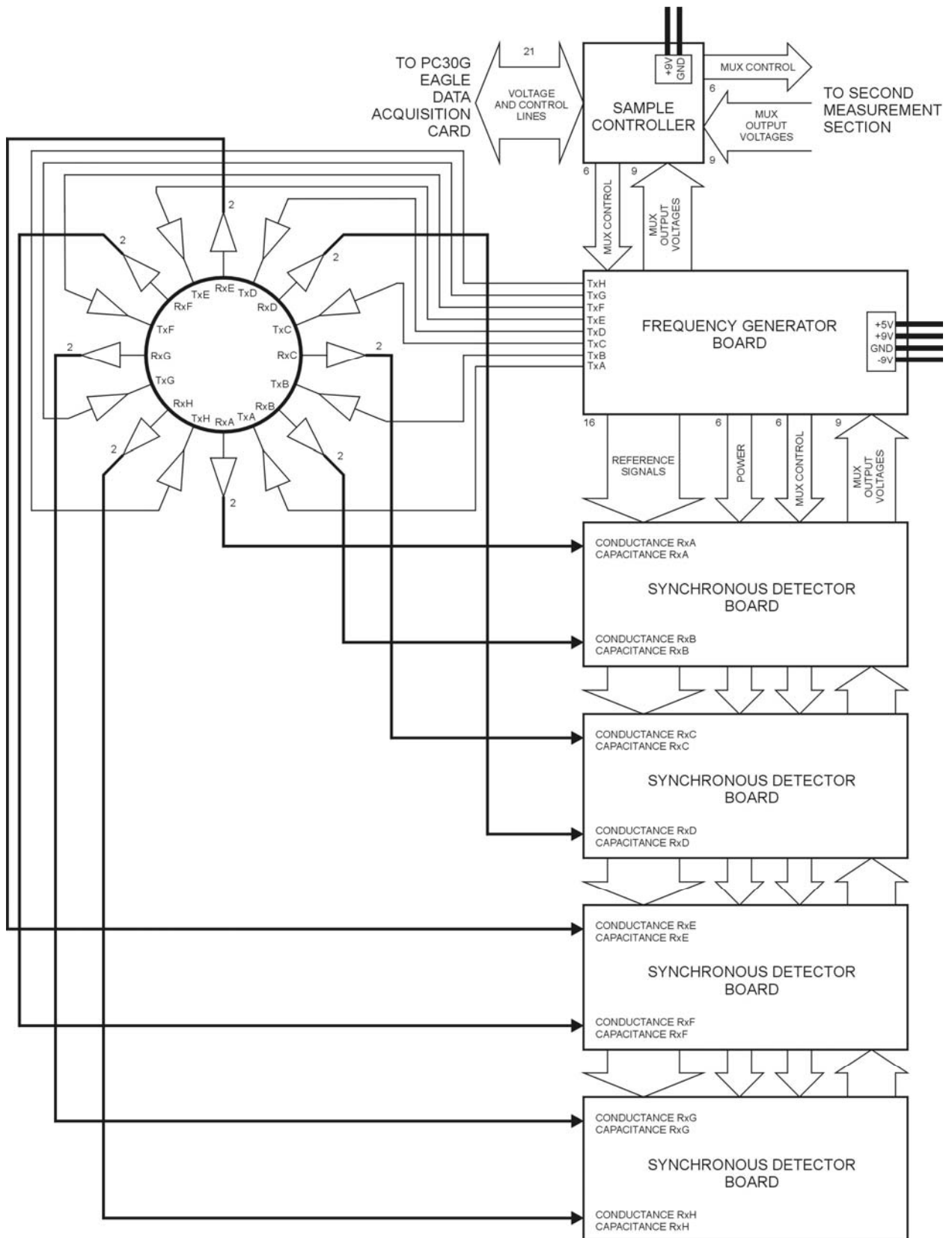


FIGURE 5.7
BLOCK DIAGRAM ILLUSTRATING THE INTERACTIONS BETWEEN THE DIFFERENT CIRCUIT BOARDS IN A 16-ELECTRODE
FREQUENCY DIVISION MULTIPLEXED IMPEDANCE TOMOGRAPHY SYSTEM, WHERE THE NUMBER NEXT TO EACH LINE
INDICATES THE NUMBER OF SIGNALS.

The operation and design of each of these modules will be explained in detail in the following sections.

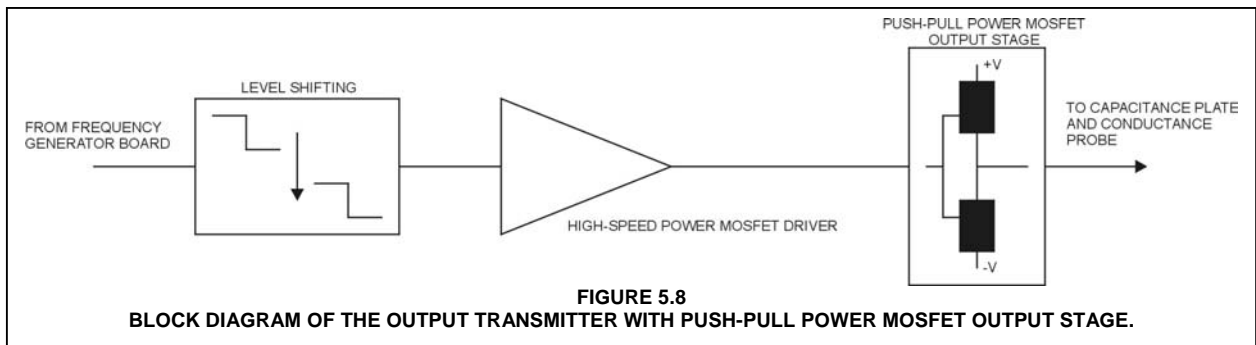
5.2.1 Output transmitter details

As mentioned previously, a LM675 power op-amp was used as the output transmitter for the 8-electrode prototype. Research was done at the Cape Technikon into the use of MOSFET drivers instead of power op-amps and a design was proposed consisting of a TC4427 dual high-speed power MOSFET driver [45]. To obtain a higher output current from the TC4427, the outputs were tied together and the inputs driven in parallel, as can be seen in Appendix C on page 122. In addition, a 10nF capacitor was placed from the output of the TC4427 to ground to minimise the output ringing and overshoot. The functionality of this capacitor can be explained by examining the output stage of the TC4427. Specifically, it consists of a push-pull MOSFET stage and these MOSFETs have a fairly high on-resistance of 10 Ω maximum. This on-resistance creates an RC combination with the capacitor thus slowing down the output rise and fall. Subsequently, the output ringing and overshoot is reduced.

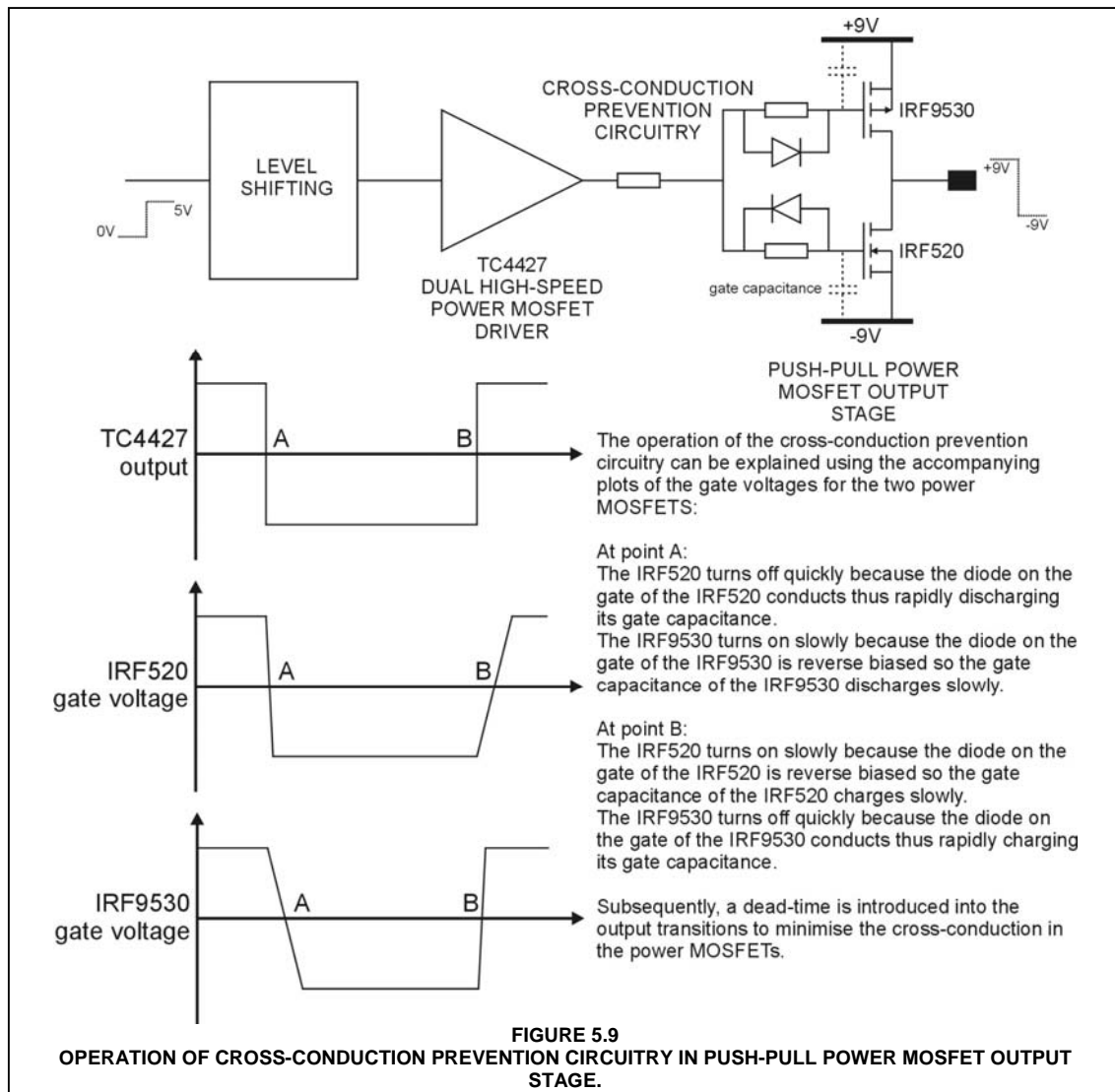
In order to generate a -9V to 9V output square wave, the ground line of the TC4427 was connected to -9V. Since the required input is a TTL signal with respect to this ground line, it was necessary to shift the input square wave from the PIC16F84 microcontroller. The potential divider configuration on the input to the TC4427 shifts the zero to 5V square wave from the microcontroller to a -9V to -4V square wave required for correct operation of the MOSFET driver.

Although good results were achieved when only a single transmitter was operating at any one time, the output waveforms were seriously degraded when more than one transmitter was operating, as would be the case for normal operation in an FDM impedance tomography system. Specifically, it was noted that superimposed on the output waveform of a particular transmitter was a signal in phase with the output from the other transmitters that were operating at the same time. This variation can also be explained with reference to the output stage of the TC4427. As mentioned previously, the on-resistance of the MOSFETs in the TC4427 is fairly high. Even with the outputs paralleled, the output impedance of the transmitter can be as high as 5 Ω . For the situation where a single transmitter is operating, the load on the transmitter is constant. Subsequently, the potential across this output impedance is constant and hence a perfect square wave is generated. When multiple transmitters are operating simultaneously, the load on a particular transmitter varies since the load on a transmitter also depends on the output states of the other transmitters in the system. Subsequently, the current drawn varies and the potential across this output impedance varies. Hence the transmitter output contains components from the other transmitters and is no longer a perfect square wave. The output impedance of the transmitter must be reduced to correct for this.

The author incorporated an additional push-pull power MOSFET output stage for each transmitter in order to achieve a lower output impedance. In particular, an IRF520 N-channel power MOSFET and an IRF9530 P-channel power MOSFET were used. These devices have a typical on-resistance of only 0.2 Ω . The TC4427 was used to drive the gates of these two MOSFETs in parallel. Since the transmitters now have a much lower output impedance, good quality square waves are generated even when all eight transmitters are operating simultaneously. Figure 5.8 on the following page is a block diagram of a transmitter.



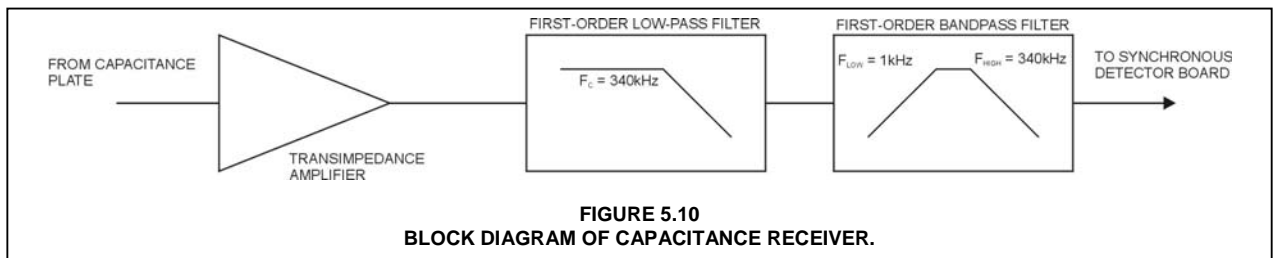
A drawback of the push-pull power MOSFET output stage is shoot-through resulting from both MOSFETs being on simultaneously, since a single MOSFET driver is being used to switch both power MOSFETs in parallel. This shoot-through resulted in ringing on the outputs, serious degradation of the power supply quality and heat dissipation in the power MOSFETs. Standard push-pull power MOSFET configurations use two independent drive signals phased so that one MOSFET is always switched off before the other is switched on. Since time and cost limitations meant that it was impractical to redesign the transmitter for a break-before-make driver, some other technique had to be used in order to generate these two independently phased drive signals from one source. The solution to this problem is illustrated in figure 5.9.



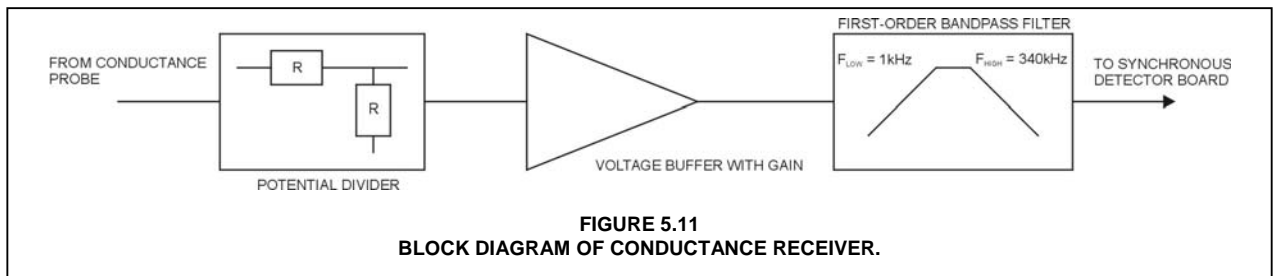
The cross-conduction prevention circuitry creates an RC time delay using the gate capacitance of the power MOSFETs. Schottky diodes are used to vary this time delay depending on whether the corresponding MOSFET is switching off or on. The operation of the cross-conduction circuitry can be explained as follows: when the TC4427 output goes low at point A, the diode on the gate of the IRF520 conducts thus rapidly discharging its gate capacitance. Subsequently, the IRF520 turns off quickly. At the same time, the diode on the gate of the IRF9530 is reverse biased so the gate capacitance of the IRF9530 discharges slowly. Subsequently, the IRF9530 turns on slowly thus introducing a dead-time between the turning off of the one MOSFET and the turning on of the second MOSFET. When the output of the TC4427 goes high the above process is reversed, and the IRF9530 turns off quickly while the IRF520 only turns on after a dead-time. Consequently, shoot-through is minimised. The circuit details of the cross-conduction circuitry, as well as the push-pull power MOSFET output stage are given in Appendix C on page 122.

5.2.2 Conductance and capacitance receiver details

Figure 5.10 is a block diagram of the capacitance receiver. It consists of a transimpedance amplifier followed by various filtering stages. The selection of the RC combination in the transimpedance amplifier follows the same design principles as discussed in the design of the 8-electrode prototype. As will become evident later, it was determined that the high frequency components of the received signals should be attenuated in order to achieve better separation at the synchronous detection stage. A first-order low-pass and bandpass filter with an upper cutoff frequency of 340kHz attenuates these high frequency components. This frequency was chosen to ensure reasonable transmission of all eight transmitter frequencies.



As before, the conductance receiver consists of a potential divider formed by an input resistor to ground and a voltage buffer. Figure 5.11 is a block diagram of the conductance receiver. A first-order bandpass filter was included to attenuate the high frequency components of the received signal and to block any DC electrochemical voltages. Electrochemical reactions between the electrodes and the conductive solution result in the generation of DC offset voltages. These DC voltages have a magnitude of up to hundreds of millivolts. A bandpass filter has been shown to reduce these offset voltages [12]. The circuit details of the conductance and capacitance receiver are provided in Appendix C on page 123.



5.2.3 Synchronous detection of receiver signals

Each synchronous detection board performs the synchronous detection of two complete channels, where a channel incorporates both the conductance and capacitance signal from a specific receiver, as can be seen in figure 5.12 on page 45.

As mentioned previously, the low-pass filtering in the 8-electrode system did not have a steep enough rolloff to prevent harmonics from the multiplication stage passing through the filter and appearing as ripple on the DC output voltages. To correct this, the 16-electrode system uses switched capacitor filters. Specifically, LMF100 dual switched capacitor filters are employed. The LMF100 can be configured to provide either a single fourth-order low-pass filter or two independent second-order low-pass filters. Due to the high component count of the 16-electrode system, it was decided that the LMF100 would be configured as two second-order low-pass filters and would perform the filtering of both the capacitance and conductance signals for each channel. An advantage of switched capacitor filters is that their cutoff frequency is set by an external clock input. Consequently, the clock inputs of the switched capacitor filters on the synchronous detector boards were tied together and were driven by the same frequency generator as transmitter G. This was an important requirement since it was not known beforehand what low-pass filter cutoff frequency would be required to achieve minimum output ripple. If standard active filters had been used then a change in the cutoff frequency would have required replacing a considerable number of capacitive and resistive components. The ratio of the cutoff frequency of the low-pass filter to the clock frequency in the current configuration is [60]

$$f_{\text{LPF}} = \frac{f_{\text{CLOCK}}}{100} \sqrt{\left(1 - \frac{1}{2Q^2}\right) + \sqrt{\left(1 - \frac{1}{2Q^2}\right)^2 + 1}} \quad (5.2)$$

where Q is the quality factor of the second-order filter and equals unity in the current configuration. A multi-turn potentiometer on the input to the LMF100 controls the gain of the switched capacitor filter and provides a means of achieving the maximum measurement range for each individual transmitter-receiver electrode pair.

One drawback of switched capacitor filters is that they are effectively a sampled-data system where the sampling rate is set by the clock input. Subsequently, the Nyquist sampling theorem must be observed and frequency components greater than one-half the clock frequency must be removed before the LMF100. To achieve this, the first-order op-amp low-pass filters were retained. The cutoff frequencies of these low-pass filters were set at 194Hz. These op-amps also provided an additional gain control stage. The gains for the op-amps in the capacitance paths were set at 2.5. The gains for the op-amps in the conductance paths were also set at 2.5 except for those corresponding to adjacent transmitter-receiver electrode pairs, where the gain was reduced to 1.2.

Another drawback of switched capacitor filters is that their outputs can be noisy. In particular, clock feedthrough results in the clock appearing as a small ripple on the output voltages of the switched capacitor filters. This was eliminated using a simple RC low-pass filter on the output of the LMF100. The cutoff frequency of this RC combination was set at 234Hz. It was decided that the convenience provided by the LMF100 to vary the cutoff frequency of all the filters in parallel far outweighed the above mentioned drawbacks. Further, since the design allowed for maximum signal range and gain, the effects of the noise introduced by the switched capacitor filters could be minimised. The circuit details of the low-pass filters can be seen in Appendix C on page 126.

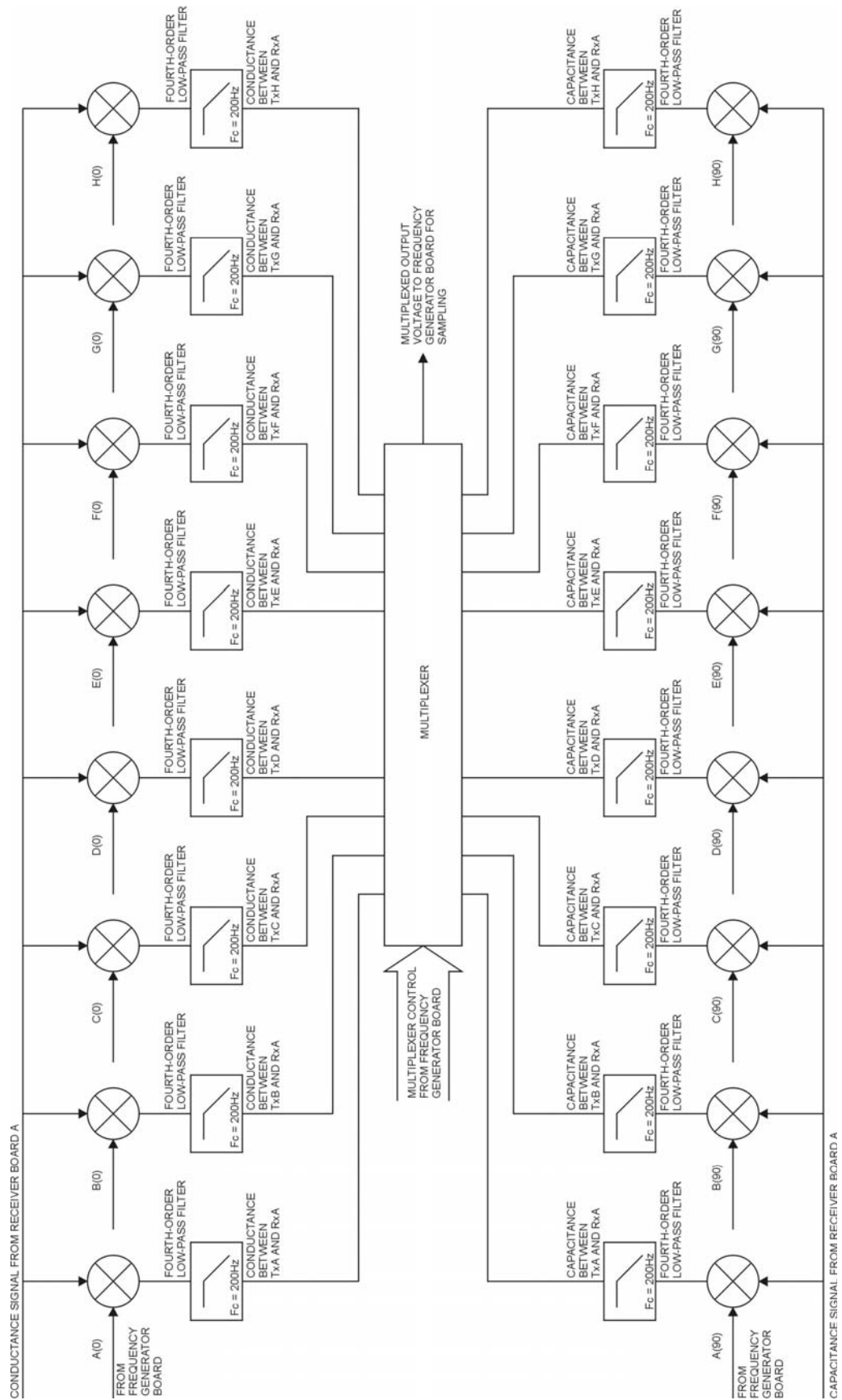


FIGURE 5.12

BLOCK DIAGRAM OF THE SYNCHRONOUS DETECTOR CIRCUIT BOARD SHOWING ONLY ONE COMPLETE CHANNEL, WHEREAS EACH SYNCHRONOUS DETECTOR CIRCUIT BOARD ACTUALLY SUPPORTS TWO COMPLETE CHANNELS. EACH MEASUREMENT PATH CONSISTS OF A DG303 CMOS ANALOGUE SWITCH TO EMULATE THE MULTIPLICATION STAGE OF THE SYNCHRONOUS DETECTION, AND A FOURTH-ORDER LOW-PASS FILTER. EACH FOURTH-ORDER FILTER IS COMPOSED OF A FIRST-ORDER OP-AMP FILTER, A SECOND-ORDER LMF100 SWITCHED CAPACITOR FILTER AND AN RC LOW-PASS FILTER ON THE OUTPUT.

As before, DG303 dual CMOS analogue switches were used to emulate the multiplication phase of the synchronous detectors. This created a problem for the quad op-amps used to generate an inverted and buffered version of the input signal to each DG303. Specifically, each DG303 has a typical input capacitance of 7pF. Since each quad op-amp drives eight DG303 switches in parallel, a capacitive load of 56pF is created resulting in op-amp instability. Replacing the standard quad op-amps with LT1356 quad op-amps from Linear Technology solved this problem. These op-amps are unity gain stable and are capable of driving capacitive loads, making them ideal for this particular application.

Multiplexing was required to capture the output voltages since the PC30G card only has 16 analogue input channels, and 128 voltages must be captured from two FDM impedance tomography systems. Unlike standard tomography systems, the multiplexing for FDM impedance tomography takes place after the measurements are performed. Consequently, it does not affect the readings and the system does not have to wait between each sample for the voltages to stabilise. In FDM impedance tomography, the multiplexing is purely for sampling purposes and it could be avoided by using a 128-input data acquisition card. In the current system, this was not a limiting factor and the multiplexing did not reduce the potential frame-rate of the FDM impedance tomography system. Consequently, only a 16-input data acquisition card was used. A single DG506A 16-input CMOS analogue switch is used to multiplex the eight capacitance and eight conductance readings for each channel. The address lines of these multiplexers are driven by the sample controller board, which will be discussed in greater detail at a later stage. The circuit details of the synchronous detector board multiplexing are provided in Appendix C on page 125.

Initial tests revealed the existence of ripple on the low-pass filter output voltages. It was determined using a spectrum analyser that the frequency of this ripple was typically within the 200Hz passband of the system. Further, it was observed that these ripple frequencies were transmitter dependent. For example, if the output of the synchronous detector for the measurement of the conductance between transmitter A and receiver A contains a 75Hz component, then this 75Hz component will also appear at the output of the synchronous detector for the measurement of the conductance between transmitter A and the other receivers. In addition, this 75Hz component will also appear at the output of the synchronous detector for the measurement of the capacitance between transmitter A and receiver A, and so on. It was also noted that this output ripple did not exist when only one transmitter was operating but would appear when two or more transmitters were operating simultaneously. Finally, it was shown that these frequency components would change if the transmitter frequencies were changed, thus proving that they were dependent on the transmitter frequencies. Further tests revealed that these frequency components were the result of the transmitter output square wave harmonics mixing in the multiplication stage of the synchronous detectors. Figure 5.13 on page 48 provides an example of the square wave harmonics mixing in the multiplication stage of the synchronous detector. Unlike the 8-electrode prototype, the frequency components generated for the 16-electrode system are within the passband of the system and therefore cannot be removed simply through the use of additional low-pass filtering. It was therefore necessary to perform a thorough analysis of these frequency components in an attempt to eliminate their generation instead of simply filtering them.

The theoretical analysis of the FDM impedance tomography system in Chapter 4 was for sine wave signals. Each transmitter only outputs a single frequency component. The output of the multiplication stage would then contain the sum and difference of these fundamental components and consequently, it is a trivial task to select appropriate transmitter frequencies to ensure that no output ripple exists within the passband of the system. However, this is not the case for a square wave FDM impedance tomography system. Specifically, each transmitter output consists of a fundamental frequency component with a number of frequency components at the odd harmonics of the fundamental frequency. In addition, if the output square waves are not perfect then minor frequency components also exist at the even harmonics of the fundamental frequency. Subsequently, the receiver signals consist of a considerably larger number of frequency components in a square wave system than a sine wave system. Since the reference for the multipliers is also a square wave, a large number of frequency components are generated at the multiplier output since the sum and difference of all the frequency harmonics in the received signal and the reference waveform are generated.

In an initial attempt to improve the situation, it was decided that the bandwidth of the receiver signals should be limited. This motivated the filtering of the high frequency components in the capacitance and conductance receivers, as discussed in the previous section. Although attenuation of these upper frequency components was achieved, they were not completely removed and subsequently, ripple was still present on the synchronous detector outputs.

The output ripple limits the accuracy of the capacitance and conductance measurement systems. Specifically, as proven in Chapter 4, the information regarding the capacitance or conductance between a particular transmitter-receiver electrode pair is incorporated in the DC component on the low-pass filter output. Any ripple superimposed on this DC component limits the accuracy with which the capacitance or conductance can be determined if only a single measurement is taken. Subsequently, the accuracy, and more importantly, the resolution of the impedance tomography system is limited. Although averaging could improve the measurement accuracy, the required frame-rate creates a time constraint resulting in only single measurements being taken.

Figure 5.13 on page 48 provides an example of a 4-electrode system illustrating the problem of the square wave harmonics mixing in the multiplication stage of the synchronous detector. Specifically, the frequency of transmitter A is 1kHz and the frequency of transmitter B is 1.81kHz. The signal at the output of receiver B consists of the frequency harmonics of both transmitters as shown. When multiplied by the reference for transmitter B, the sum and difference frequency components are generated. The frequency components below 1kHz are shown in figure 5.13. Specifically, the 50Hz component is the difference between the 9th and 5th harmonic of transmitter A and transmitter B respectively, the 430Hz component is the difference between the 5th and 3rd harmonic of transmitter A and transmitter B respectively and the 810Hz component is the difference between the fundamental frequencies of both transmitters. Since the 50Hz component is within the passband of the system, it will not be filtered and will appear at the synchronous detector output as ripple. Although this is a manufactured example, it does illustrate the complication of using square waves in a FDM impedance tomography system. It is also shown in figure 5.13 that if the bandwidth of the received signal was limited to approximately 8kHz, then there would be no output ripple, if an ideal low-pass filter was used.

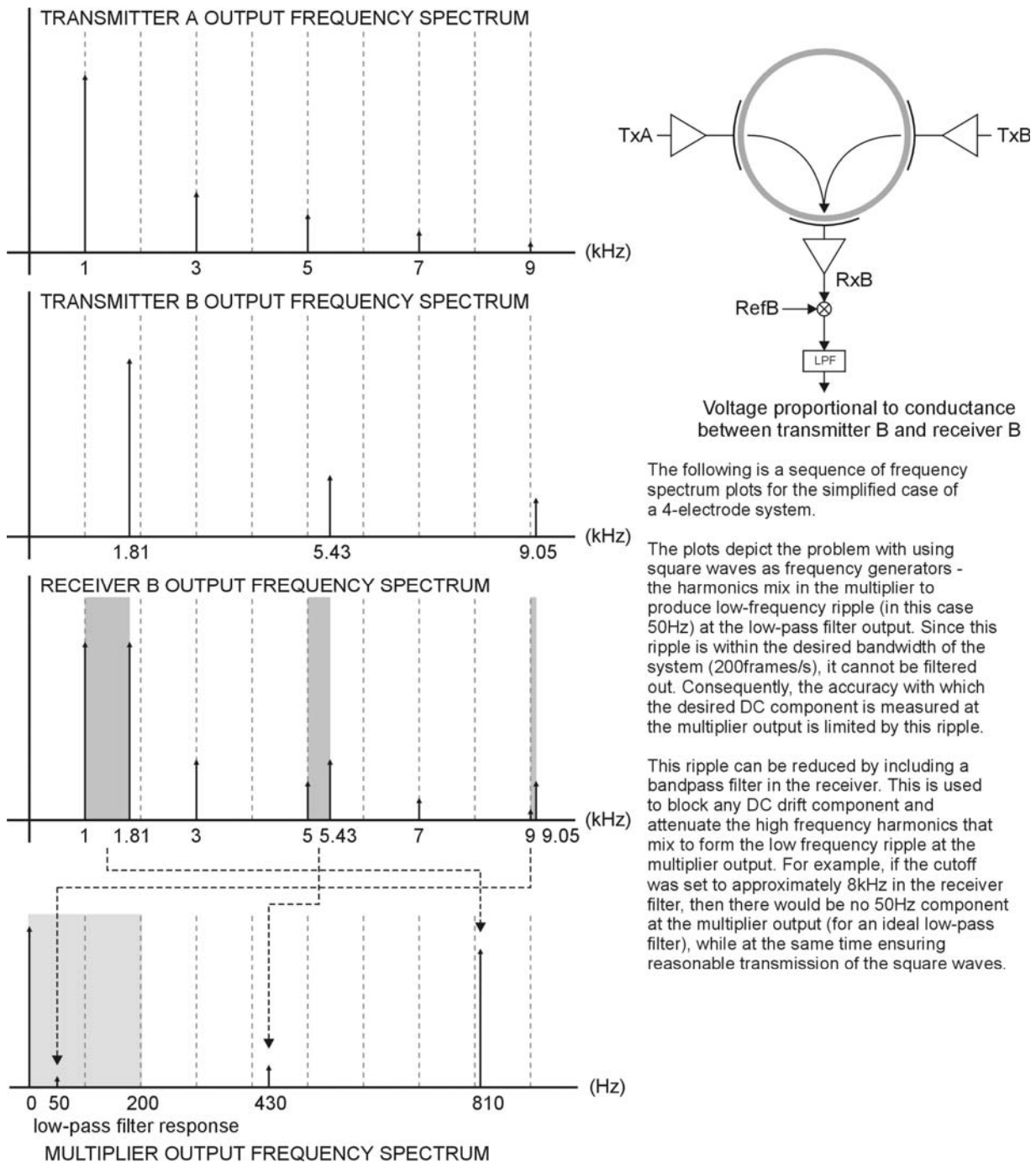


FIGURE 5.13
EXAMPLE OF THE SQUARE WAVE HARMONICS MIXING IN THE MULTIPLICATION STAGE OF THE SYNCHRONOUS DETECTOR FOR A 4-ELECTRODE SYSTEM. NOTE THAT THE HORIZONTAL SCALE OF THE MULTIPLIER OUTPUT FREQUENCY SPECTRUM IS DIFFERENT TO THAT OF THE TRANSMITTER AND RECEIVER FREQUENCY SPECTRUMS.

It was decided that a better understanding of these frequency-mixing principles could be obtained through the simulation of the system operation in software. Using the specified eight transmitter frequencies, the simulator calculates the multiplier output frequencies of each synchronous detector in a channel. Since the output frequency components are independent of the particular receiver, the simulation is only performed for a single receiver. Initially, the magnitudes of the frequency components within the system were not incorporated. Instead, frequencies higher than some maximum threshold were simply ignored. Although the initial results of this simulation were promising, it would fail to predict the generation of certain frequency components resulting from the mixing of those higher frequency harmonics that had been ignored. It was therefore necessary to include the magnitude of the frequency components in the simulation to ensure that all possible output frequency components are correctly predicted for the specified set of transmitter frequencies.

The program code for the modified simulator is included in Appendix J on page 138 and operates as follows:

1. Calculate the magnitude of the received frequency components. All even and odd harmonics up to and including the 100th harmonic of the fundamental frequency for all eight transmitters are incorporated, but only those harmonics whose magnitudes are greater than some minimum threshold are included in the later stages of the simulation. Consequently, the attenuation of the low-pass and bandpass filters in the capacitance and conductance receivers are incorporated in this calculation. Specifically, the initial magnitude of harmonic 'c' is calculated, using the Fourier transform of a unity magnitude square wave, as $\frac{1}{\pi c}$. For a fundamental frequency 'f', the gain of the low-pass filter for harmonic 'c' is calculated as $\frac{1}{\sqrt{1 + (2\pi f c R_{LPF} C_{LPF})^2}}$. The gain of the bandpass filter is calculated in a similar way. The final magnitude for harmonic 'c' is calculated as $\text{gain}_{LPF} \times \text{gain}_{BPF} \times \frac{1}{\pi c}$ and is only included in the later stages of the simulation if this magnitude is greater than the specified threshold. For completeness, a DC component is included in the received signal to compensate for any DC offsets resulting from electrochemical reactions, component offsets and so on.
2. Sort the remaining received frequencies after thresholding and check for duplication. Specifically, if duplicate frequency components are generated then the minimum frequency separation can immediately be calculated as 0Hz. Duplicate frequency components indicate that the frequencies of two or more transmitters are related, and is an undesirable situation because of the following: consider the case of the 4-electrode system with transmitter A operating at 3kHz and transmitter B operating at 5kHz. In this example, the 5th harmonic of transmitter A equals the 3rd harmonic of transmitter B and duplicate frequencies would be received. To calculate, for example, the conductance between transmitter A and receiver A, the received signal is multiplied by reference A at 3kHz. However, transmitter B contains a harmonic whose frequency equals a harmonic of transmitter A and will therefore pass through the low-pass filter stage of the synchronous detector. Consequently, the output of the low-pass filter will not only represent the conductance between transmitter A and receiver A but will also incorporate the conductance between transmitter B and receiver A, which is clearly an undesirable situation. In terms of output ripple, it is also undesirable to have duplicate frequency components. Due to component tolerances, the duplicate frequency components will not be exactly equal in a practical implementation and instead a low frequency difference will exist. This difference frequency will typically be within the passband of the system and will appear as ripple on the low-pass filter outputs.

3. Calculate the harmonics of the reference square waves for all eight transmitter frequencies. Since the DG303 uses TTL-compatible input levels, the reference waveforms are assumed to be perfect square waves and hence only odd harmonics are included. All odd harmonics, up to and including the 100th harmonic of the reference frequency are included.
4. Calculate the sum and difference frequencies at the multiplier outputs for each of the eight transmitter frequencies. With each transmitter as a reference frequency, the sum and difference frequency components between all the received frequency components and the harmonics of the reference square wave are calculated.
5. Simulate the aliasing in the switched capacitor filters. Since the LMF100 is a sampled-data system, any frequency components greater than one-half the clock frequency result in aliasing. In this application, the clock frequency equals the frequency of transmitter G. The effect of the aliasing is to shift frequency components from $\frac{f_{\text{CLOCK}}}{2} + X$, where X is the difference between the input frequency and $\frac{f_{\text{CLOCK}}}{2}$, down to $\frac{f_{\text{CLOCK}}}{2} - X$. Hence, components in the region of the clock frequency are shifted down into the passband of the switched capacitor low-pass filter and appear at the output as ripple. For example, if the clock frequency was 20kHz and the multiplier output contained a frequency component at 19.9kHz, this frequency component would be shifted down to 100Hz and would appear on the switched capacitor filter output as a 100Hz ripple component.

Initial tests using this simulator showed that it is undesirable to have transmitter frequencies that are related. Initially, the eight transmitter frequencies were generated using two PIC16F84 microcontrollers using the same software as the 8-electrode prototype. Simply by using two different crystal frequencies, eight different output frequencies are generated. This is discussed in greater detail in section 5.2.4. As mentioned in section 4.2.1 on page 25, the periods of the square waves generated by each microcontroller are 8, 12, 16 and 20 cycles respectively. Consequently, the second harmonic of the output whose period is 16 cycles equals the fundamental frequency of the output whose period is 8 cycles. This would not be a problem if perfect square waves were generated and propagated throughout the system. However, frequency components are generated at the even harmonics of the fundamental frequency because of imperfections in the transmitters and the limited slew rate of the receiver op-amps. Since all four frequencies from each microcontroller are related by the crystal frequency of that microcontroller, it is impossible to generate eight frequencies whose harmonics do not coincide using this technique. Subsequently, it was decided that each transmitter should be driven by its own independent frequency generator source. In addition, since each frequency is generated independently of the other frequencies, it should be possible to determine a set of eight frequencies that would result in minimal output ripple. Specifically, an optimisation search algorithm would be used in conjunction with the simulator to determine the set of eight output frequencies that produce optimal frequency separation and hence minimal output ripple.

5.2.4 Population-based incremental learning fundamentals

Population-based incremental learning (PBIL) is a stochastic search technique that combines the mechanisms of genetic algorithms with competitive learning to produce an optimisation tool that is simpler than genetic algorithms and which out-performs genetic algorithms both in terms of speed and accuracy for a large set of optimisation problems [61]. Genetic algorithms are biologically motivated adaptive systems that are based on the principles of natural selection and genetic recombination [61]. The operation of a genetic algorithm to solve an optimisation problem can be explained as follows: genetic algorithms maintain a population of trial solutions to a particular optimisation problem. These trial solutions are initially randomly generated. For each generation, these trial solutions are evaluated as to their fitness, which is a measure of how well a trial solution optimises the objective function [61]. Trial solutions that produce good results are then combined using various recombination operators to generate a new generation of potential solutions. In addition, genetic diversity must be maintained to ensure that the algorithm performs a thorough search of the feature space and does not converge too quickly on a sub-optimal solution. This is achieved through mutation. Mutation helps preserve diversity by introducing random changes into the population [61]. By iterating the above process over a number of generations, the populations of trial solutions produced move steadily towards regions of high performance, although it is not guaranteed that the optimum solution will be found.

The PBIL algorithm attempts to create a single probability vector from which trial solutions are drawn to produce the next generation [61]. Each trial solution is encoded as a binary sequence of '1' or '0'. The probability vector represents the probability of each bit position containing a '1' [61]. At the start of the algorithm, the values of the probability vector are initialised to 0.5 representing an equal probability of generating a '1' or '0' in that particular bit position. Consequently, the trial solutions generated are a random sequence of '1' and '0'. For each generation, the PBIL algorithm actively updates this probability vector to represent high evaluation trial solutions. This is achieved through gradually shifting the values in the probability vector away from 0.5 and towards the bit values of the best trial solution for each generation. The probability update rule is described below [61]

$$PV_j = (PV_j \times (1 - LR)) + (LR \times TS_j) \quad (5.3)$$

where PV is the probability vector and PV_j is the probability that bit position 'j' contains a '1', LR is the learning rate and TS is the trial solution towards which the probability vector is being shifted. Consequently, the values of the probability vector are steadily shifted away from 0.5 and towards either 1.0 or 0.0, thus defining a single high evaluation trial solution.

A mutation operator is also provided to ensure that the algorithm does not converge too quickly on a sub-optimal solution. Mutation in PBIL can either be performed on the trial solutions themselves or on the probability vector [61]. The bits to be mutated in the probability vector are chosen randomly according to the mutation probability value and are updated according to the following equation [61]

$$PV_j = PV_j \times (1 - \text{SHIFT}) + \text{random}[0 \text{ or } 1] \times \text{SHIFT} \quad (5.4)$$

where SHIFT controls the amount of mutation to apply to the probability vector.

Overall, the operation of the PBIL optimisation algorithm can be described using the following pseudo-code:

```

for all j do
    initialise  $PV_j$  to 0.5
for generations do
    for population do
        generate trial solution using PV
        evaluate fitness of trial solution
        if fitness better than best trial solution TS fitness then
            update record of best trial solution TS
    for all j do
        adjust PV towards TS using the probability update rule  $PV_j = (PV_j \times (1 - LR)) + (LR \times TS_j)$ 
        if mutation is to be applied to bit j then
            mutate probability vector according to the mutation update rule  $PV_j = PV_j \times (1 - \text{SHIFT}) + \text{random}[0 \text{ or } 1] \times \text{SHIFT}$ 

```

As is evident above, the PBIL algorithm requires the setting of four parameters, namely the population size, the learning rate, the mutation probability and the mutation shift [61]. Since the probability vector is used to generate the next population of trial solutions, the learning rate has a direct affect on the regions of the function space that will be explored [61]. If the learning rate is too high, then the initial populations generated largely determine the focus of the search and a thorough exploration of the function space is not achieved [61]. However, if the learning rate is too low then it could take an enormous number of generations to shift the probability vector towards the high evaluation regions and escape from initial oscillation [61].

In terms of the intended application, the PBIL algorithm is used to determine the set of eight transmitter frequencies that maximise the frequency separation between the harmonics of these eight frequencies. The accompanying program code can be found in Appendix J on page 140. The results of the PBIL optimisation algorithm will be discussed in the next section once the generation of the transmitter frequencies has been examined in greater detail.

TABLE 5.1
TRANSMITTER FREQUENCIES GENERATED USING TWO
PIC16F84 MICROCONTROLLERS WITH 4.43361MHz AND
5MHz CRYSTALS RESPECTIVELY.

5 MHz CRYSTAL	
TRANSMITTER	OUTPUT FREQUENCY
A	78.125kHz
B	52.0833kHz
C	31.25kHz
D	39.0625kHz
4.43361 MHz CRYSTAL	
TRANSMITTER	OUTPUT FREQUENCY
E	69.2752kHz
F	46.1834kHz
G	27.7101kHz
H	34.6376kHz

5.2.5 Generation of transmitter frequencies

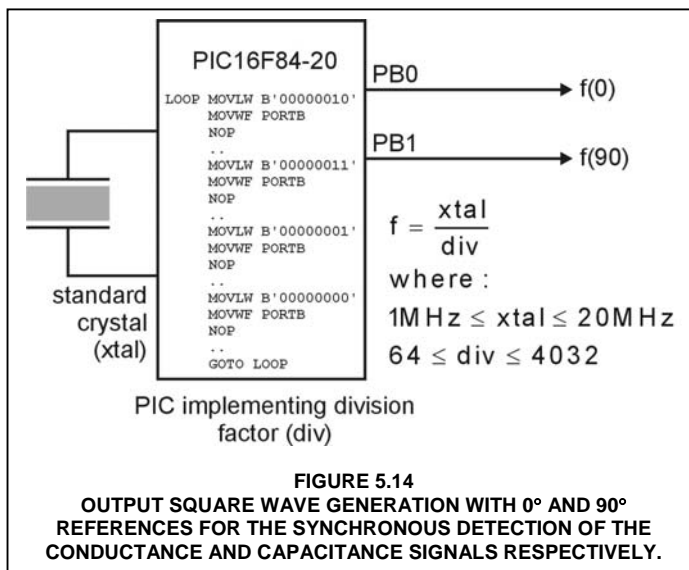
As mentioned previously, the generation of the eight transmitter frequencies was initially achieved using two PIC16F84 microcontrollers programmed using the same square wave generation software as was used for the 8-electrode system. Simply by using two different crystal frequencies, eight output square waves and their quadrature waveforms were generated with the correct duty cycle and phase shift. The crystal frequencies chosen were 4.43361MHz and 5MHz and the output frequencies generated are given

in table 5.1. Transmitter G was chosen as the lowest output frequency since it was used as the clock input to the switched capacitor low-pass filters. Subsequently, the cutoff frequencies of the switched capacitor filters were 352Hz.

It is impossible to generate eight independent output frequencies using this technique, since all four frequencies for each microcontroller are related by the crystal frequency. In order to generate independent frequencies, a different frequency generation technique would be required. Initially, CMOS 555 oscillators were investigated. However, tests revealed that the output frequency would drift over time since the timing of the oscillator depended on an RC combination. To achieve the required frequency stability, it was necessary to employ a crystal-based frequency generator. Although a number of logic gate based designs were tested, it was decided that a single PIC16F84A microcontroller would be used as each frequency generator. This may seem an overly elaborate way of generating a square wave and its quadrature, but the microcontroller implementation has certain advantages compared to the logic gate based designs, namely

- Component count, since each output square wave and its quadrature are generated using a single 18-pin integrated circuit.
- Versatility, since the output frequency generated can easily be modified by reprogramming the microcontroller.

The program code implemented on the microcontroller is similar to that used for the 8-electrode prototype. However, only one square wave is generated and therefore a sequence of only four port set instructions is required to generate the output square wave and its quadrature. Between each of the four port set instructions are a number of 'nop' instructions. The first three port set instructions are followed by a minimum of two 'nop' instructions. The 'goto' instruction after the fourth port set instruction requires two instruction cycles to execute hence the 'nop' instructions ensure the correct duty cycle and phase shift. Additional 'nop' instructions can also be added equally after all four port set instructions to decrease the output frequency for a given crystal frequency.



The program code is essentially implementing a division of the crystal frequency, where the minimum division factor is 64 corresponding to no additional 'nop' instructions. The maximum division factor is limited to 4032 only by the available program memory on the PIC16F84A. Figure 5.14 illustrates these concepts. A program written in Delphi generates the assembler code used to achieve a specific division factor automatically. The user specifies the desired division factor and this program generates a text file that is assembled using MPASM and then downloaded to the corresponding microcontroller. The program code for this Delphi application can

be found in Appendix I on page 137. In addition, the assembler code generated for a division factor of 64 is included in Appendix F on page 134.

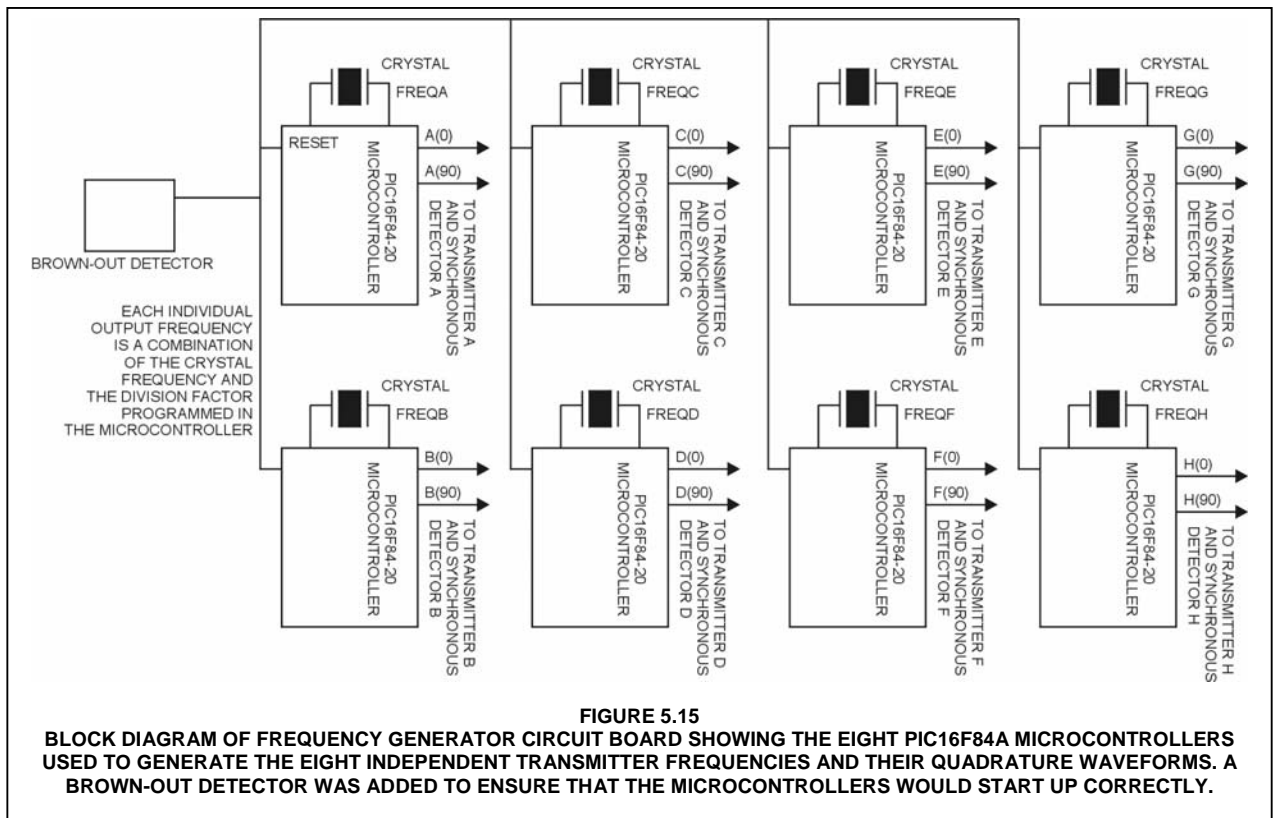


Figure 5.15 is a block diagram of the frequency generator board. A brown-out detector was included to ensure that all eight microcontrollers start up correctly. Initial tests revealed that certain transmitters were not operating correctly after switch-on. Further tests revealed that a brown-out condition was occurring during start up. Subsequently, certain microcontrollers would enter an undefined state resulting in the corresponding transmitters not operating correctly. The addition of the brown-out detector solved this problem. The circuit details of the frequency generator board are provided in Appendix C on page 124.

Having designed the frequency generator board, it was necessary to select eight frequencies that maximise the frequency separation of the harmonics. As mentioned previously, the PBIL optimisation algorithm will be used to perform this task. The implementation of the PBIL algorithm to this specific application will be discussed briefly in the following section.

Firstly, the algorithm requires a set of possible transmitter frequencies from which it can select a subset of eight frequencies as a trial solution. By combining the set of standard crystal frequencies available with all the possible division factors that the PIC16F84A microcontroller can implement, a set of 256 different frequencies were generated between 17.6kHz and 80kHz. The index of each frequency is coded as an 8-bit value. Subsequently, a trial solution is a 64-bit binary sequence where each byte indexes into this array to select the corresponding transmitter frequency. Secondly, the simulation algorithm is modified to find the minimum ripple frequency on the output of the synchronous detectors for a given set of eight transmitter frequencies. This corresponds to the fitness of a trial solution. If this minimum ripple frequency is maximised then the output ripple will be minimised. In this way, the algorithm selects a set of eight transmitter frequencies that minimise the output ripple. The accompanying program code can be found in Appendix J on page 138.

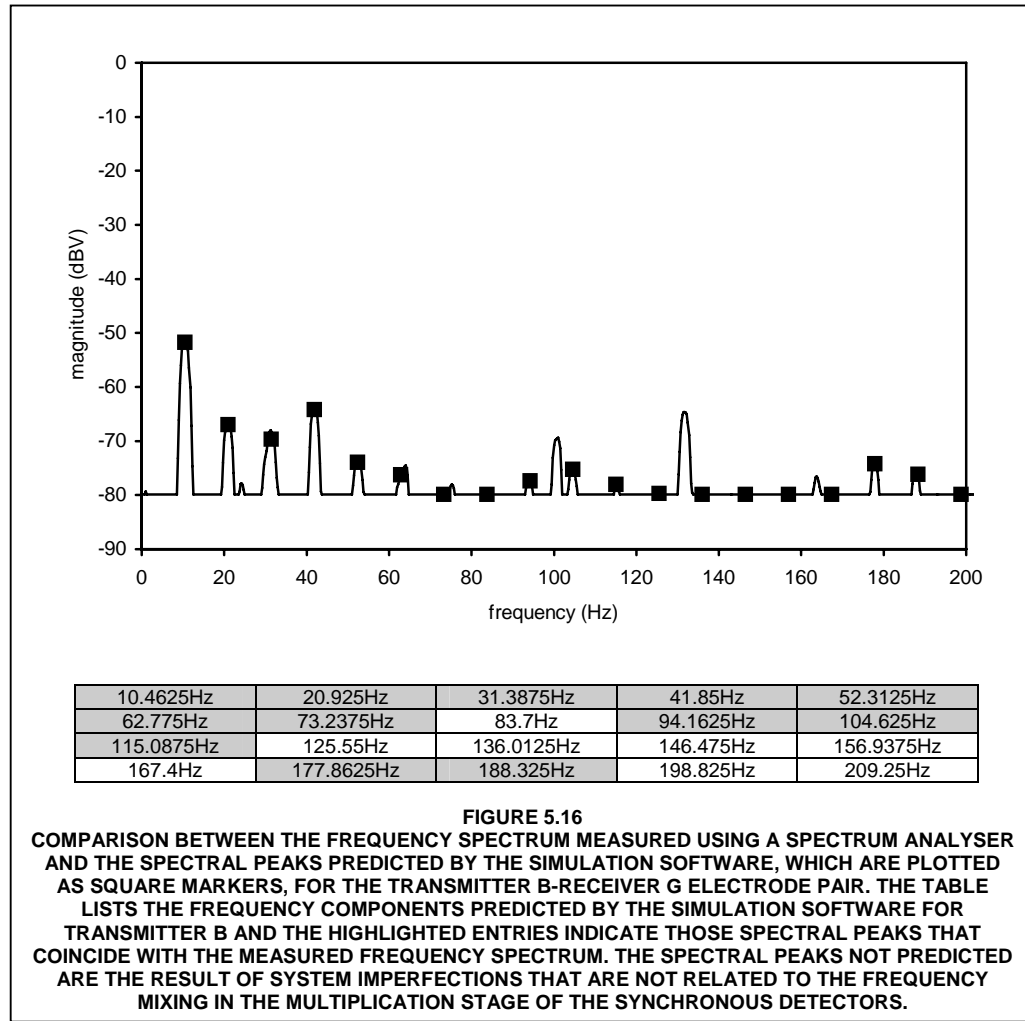
The only required user input is the selection of the magnitude threshold that determines which frequency harmonics to consider. Due to the simplicity of the simulation used to model the system behaviour, this magnitude threshold cannot be equated to a measurable signal level and therefore does not have a unit. Instead, it is found through experimentation. By observing the frequency components on the output of the low-pass filters with a spectrum analyser, it was possible to verify the simulation results. Specifically, tests revealed that for a threshold value of 0.0001, all frequency components observed with the spectrum analyser were successfully predicted by the simulation software. However, the best result achieved using the PBIL optimisation algorithm for this threshold value was a minimum frequency component at 35Hz on the low-pass filter outputs. Since this is within the passband of the system, output ripple would be observed. By increasing the threshold value to 0.01, the minimum frequency component increased but not all the frequency components were predicted. Subsequently, it was decided that a compromise should be reached by using a threshold value of 0.01. Specifically, all the major frequency components will be detected and should be outside the passband of the low-pass filters. The drawback is that minor frequency components may still exist on the low-pass filter outputs that were not predicted by the simulation software. Using this threshold, the PBIL algorithm was able to determine a set of eight transmitter frequencies with a minimum frequency component at 829Hz, which should be filtered out by the fourth-order low-pass filter stages. It was observed that the performance of the PBIL algorithm varied on each run resulting from the convergence on local optima instead of an absolute optimal solution. Subsequently, the above tests were repeated on multiple occasions.

The eight output frequencies selected using the PBIL optimisation algorithm are detailed in table 5.2, which includes the corresponding crystal frequencies and division factors required to generate these frequencies. Further, Appendix H on page 136 lists up to the 9th harmonic of these fundamental frequencies.

TABLE 5.2
TRANSMITTER FREQUENCIES SELECTED USING PBIL OPTIMISATION ALGORITHM WITH THE CRYSTAL FREQUENCIES AND DIVISION FACTORS REQUIRED TO GENERATE THESE FREQUENCIES.

TRANSMITTER	OUTPUT FREQUENCY	REQUIRED CRYSTAL FREQUENCY	REQUIRED PIC DIVISION FACTOR
A	79.1718kHz	8.867238MHz	112
B	75kHz	12MHz	160
C	73.5294kHz	20MHz	272
D	66.6667kHz	16MHz	240
E	62.8364kHz	11.0592MHz	176
F	54.3478kHz	20MHz	368
G (also LMF100 clock)	23.4375kHz	12MHz	512
H	48kHz	6.144MHz	128

To assess the accuracy of the simulation, the above transmitter frequencies were implemented and the frequency components on the outputs of the low-pass filters were monitored using a spectrum analyser. As an example, figure 5.16 on page 56 contains the frequency spectrum captured on the output of the low-pass filter for the measurement of the conductance between transmitter B and receiver G. Included in figure 5.16 is a table of the frequency components predicted for this particular transmitter-receiver electrode pair using the simulation software. The highlighted entries indicate those frequency components observed with the spectrum analyser. The predicted frequency components are displayed on the frequency spectrum as square markers.



As can be seen in figure 5.16, the simulation software is able to predict a large percentage of the actual frequency components measured. The simulation software has shown that it is impossible to obtain a set of eight transmitter frequencies for which there is no output ripple in a square wave FDM impedance tomography system. In a pure sine wave system, it is a simple task to choose eight output frequencies for which there will be no output ripple since there are no frequency harmonics to consider. A sine wave FDM impedance tomography system is currently being developed at the University of Cape Town for the 8-electrode prototype rig to verify this assumption [62]. In theory, the sine wave system is expected to produce more accurate results at a higher image resolution. However, a drawback of the sine wave system is cost, since analogue multipliers, which are considerably more expensive than the DG303 CMOS switches used in the current system, are required to perform the multiplication. It is also more complex and expensive to generate a sine wave and its quadrature waveform. All tests in the following sections are conducted with the 16-electrode square wave FDM impedance tomography system.

5.3 Software Design of a 16-electrode Frequency Division Multiplexed Impedance Tomography System

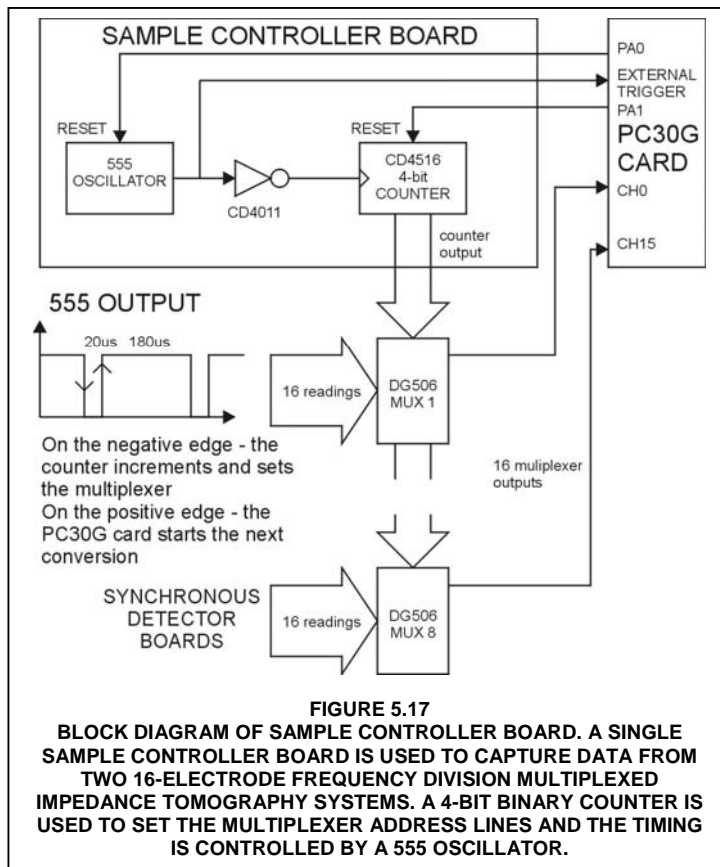
The following sections examine the software developed for the dual-plane 16-electrode FDM impedance tomography system.

5.3.1 Data capture software description

As mentioned previously, the polled sampling technique used for the 8-electrode impedance tomography system is not fast enough to achieve the desired frame-rate. In addition, the maximum data capture rate can only be achieved if DMA is used. Specifically, since 128 voltages must be captured from two systems at a rate of 200frames/s, the desired sampling rate is 51.2ksamples/s. Therefore it was necessary to modify the data capture software to use streaming so that the maximum frame-rate could be achieved. Streaming is similar to single channel DMA sampling in that it only requires a single DMA channel. However, streaming allows for unlimited amounts of data to be transferred through the use of a circular buffer [63]. All the register-level instructions for programming the Eagle data acquisition cards are incorporated in the EDR software development kit provided with the card. Specifically, this functionality is accessed by linking the associated dynamic link library (DLL) device driver into a program at run time. These high-level procedures then perform the register-level operations to achieve a specific task.

An additional complication of the current system is that the analogue multiplexers have to be configured for each set of 16 voltage readings. Initially, the address lines of these multiplexers were to be driven by the digital input-output lines on the PC30G card, as was done for the 8-electrode system. However, these port values must be set through software. Consequently, any advantages that may have been gained through the use of streaming are negated by the fact that every 16 samples the software has to increment the address lines of the multiplexers for the next set of 16 readings. As expected, the use of DMA only results in a performance increase if a large number of samples can be captured without software intervention. Some other technique of setting the multiplexer address lines was therefore required.

It was decided that faster sampling could be achieved if the address lines of the multiplexers were driven by external hardware, and figure 5.17 on page 58 is a block diagram of this sample controller board. A single sample controller board is used to drive both impedance tomography systems and is configured using the digital input-output lines of the PC30G card. In contrast to the previous system, these digital lines are only used to configure the sample controller board before and after a complete set of data has been captured and therefore do not affect the sampling rate of the system.



Central to the sample controller board is a 4-bit binary counter that is used to drive the address lines of all the multiplexers in parallel. A CMOS 555 oscillator provides the clock signal to both the 4-bit counter and the external trigger input of the PC30G card. A timing diagram of the 555 output is also included in figure 5.17 and the operation can be explained as follows: on the negative transition of the 555 output, the 4-bit binary counter is incremented and the multiplexers connect the next voltage inputs in the sequence to the analogue input channels of the PC30G card based on the address lines. The 555 output is low for a period of 20us while the multiplexer outputs stabilise. Specifically, the maximum switching time of a DG506A CMOS analogue multiplexer is 1us. On the positive transition of the 555 output, the analogue-to-digital converter on the PC30G card is triggered and the process of

sampling the 16 analogue inputs is started. The 555 output is held high for a period of 180us for this sampling to complete. Specifically, since the maximum sampling rate of the card is 100ksamples/s, the minimum time taken to capture 16 samples is 160us. In total, the time taken to configure the multiplexers and capture 16 voltage readings is 200us corresponding to a potential sampling rate of 80ksamples/s. Since the required sampling rate is only 51.2ksamples/s, it is clear that the addition of a sample controller board achieves the desired data capture rate. The circuit details of this sample controller board are included in Appendix C on page 128.

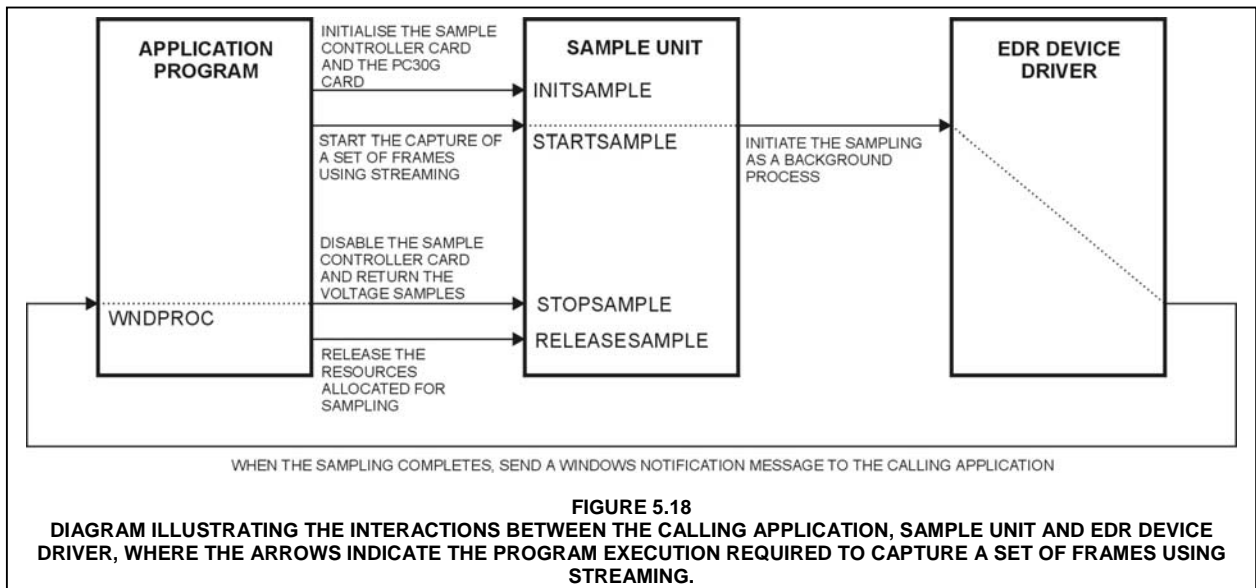
In summary, the operation of the sample controller board is as follows:

1. digital output PA0 is held low to keep the 555 in the reset state
2. digital output PA1 is held high to reset the 4-bit counter to '0000'
3. digital output PA1 returns to the low state
4. sampling is started by outputting a high on PA0 thus releasing the 555 from the reset state
5. since the multiplexers are already configured for address zero, the positive transition of the 555 output results in the voltages on input 0 of all 16 multiplexers being sampled by the PC30G card
6. on the negative transition, the counter increments to '0001' and the multiplexers connect the voltages on input 1 to the analogue input channels of the PC30G card
7. on the positive transition, these 16 voltages are sampled by the PC30G card
8. this process repeats until the counter overflows to '0000' corresponding to the collection of a complete frame of 128 voltages from both impedance tomography systems
9. sampling is disabled by outputting a low on PA0 thus keeping the 555 in the reset state

Having designed a hardware system for automatically controlling the multiplexer address lines, it was still necessary to implement a software unit that will provide high-level access to the sampling functions. This unit is included in Appendix K on page 149. It was determined that greater efficiency would be achieved by capturing the data in sets of frames, instead of individual frames. Although the sampling unit allows up to 200 frames to be captured at one time without software intervention, tests have shown that readings for the later frames, namely those above frame number 100, can be erroneous as a result of a temporary loss of synchronisation. As is evident from the description of the sample controller board, exact synchronisation is required between the hardware controlling the multiplexers and the PC30G card in order to know that, for example, sampled voltage number 1 corresponds to the capacitance between transmitter A and receiver A. If noise on the external trigger line initiates a sample that is not related to the multiplexer settings then all samples captured from that point are out of synchronisation by the 16 additional samples captured. Only by restarting the data capture process will synchronisation once again be achieved. Since this technique permits no means of verifying the synchronisation between the hardware and the PC30G card, it is necessary to ensure that such a loss of synchronisation does not occur. Tests have shown that sufficiently reliable results can be achieved if the maximum number of frames captured without software intervention is limited to 50. Consequently, all further data capture software will capture readings in sets of 50 frames so as to maximise the efficiency introduced through the use of streaming, while at the same time minimise the possibility of a loss of synchronisation.

The operation of the sampling unit can be explained with reference to figure 5.18 on page 60. When the program starts, the sampling unit and the PC30G card are initialised by calling 'InitSample'. This procedure configures the sample controller board by resetting the 4-bit counter and holding the 555 in the reset state. It then proceeds to configure the PC30G data acquisition card as follows:

- the channel list represents the sequence of analogue input channels that must be sampled when a trigger is received. InitSample loads channel 0 to 15 into the channel list and sets the gain of each channel to unity.
- burst mode is a feature provided by the data acquisition card where for every trigger received, a specific number of channels are sampled at the maximum sampling rate of the card, instead of just a single conversion. InitSample configures the PC30G card to use burst mode so that for every external trigger received, all 16 analogue input channels are sampled at the maximum rate permitted.
- as mentioned previously, streaming transfers the sampled data directly to memory. InitSample instructs the PC30G card to use streaming and that the data should be transferred to memory in blocks of 1024 samples, corresponding to four complete frames of data from both systems. The larger the transfer block, the more efficient the streaming. A drawback is that samples can then only be captured in multiples of the transfer block size. Consequently, if only one frame were required from both systems, an additional 768 samples would also be captured and then discarded. The choice of the transfer block size is a compromise between these two factors.
- finally, InitSample configures the EDR driver so that when the DMA transfer is complete, a Windows notification message is sent by the driver to the application program. InitSample then returns the message identification value for this notification message to the calling application.



Calling 'StartSample' initiates the data capture process. The sampling unit uses a buffer to store the results from the streaming. It instructs the EDR device driver to begin sampling as a background process and specifies the number of samples to be captured. It also provides the device driver with the address in memory of the buffer as the desired location for the samples. Sampling starts by releasing the 555 from the reset state, as discussed before, and the program execution returns to the calling application. From this point on, the sampling is completely controlled by the EDR device driver as a background process. Consequently, the calling application can proceed with some completely unrelated task while the device driver captures the next set of frames. As will be seen when real-time on-line reconstructions are examined, this provides a convenient time for the image reconstructions to be performed for the previous set of captured frames.

Once the required number of frames has been captured, the EDR device driver posts a Windows notification message to the calling application. To ensure that this message is captured by the calling application, it is necessary to override the default Windows message handling procedure 'WndProc'. Essentially, if the message relates to some standard Windows functionality then the original 'WndProc' procedure is called. However, if the message identification value matches that returned by 'InitSample' then the sampling is complete and 'StopSample' is called. 'StopSample' returns the 555 to the reset state and converts the raw binary data stored in the buffer into actual voltage readings in the range from $-5V$ to $5V$. These voltage readings are then returned to the calling application. This concludes the capture of the 50 frames. When the program exits, 'ReleaseSample' must be called to de-allocate the resources reserved for sampling. Tests were conducted to determine the on-line data capture rate of the sampling unit. Although the sample controller board is designed to capture data at 80ksamples/s, corresponding to 312frames/s from both systems, the overhead of processing and storing of results reduced the on-line frame-rate to approximately 280frames/s. However, this still meets the required specification of 200frames/s from both systems.

5.3.2 Training and test database generation software description

As mentioned previously, it is necessary to capture a large quantity of representative data in order to train a neural network. A data capture application was developed for the task of generating this training database. The complete program code for this application is provided in Appendix K on page 142 with comments discussing the functionality of each of its procedures. Although the fundamental concepts of this program are the same as those used for previous research, various modifications were made in an attempt to reduce the time taken to generate a complete training database. An example of such a modification was the extensive use of the registry to save the program configuration. Although the time taken to configure the program for an individual test may not be that significant, the advantage achieved through saving the program state is definitely observed when the same or similar test must be repeated.

The training and test data are stored in databases instead of individual text files. This has the obvious advantage of encapsulating all the training or test data results into a minimum number of files. In addition, the use of databases simplifies the task of accessing a particular test point or number of test points and speeds up the loading of complete databases into program memory. The software used to add entries to the database is also more flexible through the use of databases. It is possible to edit entries, overwrite entries or completely remove entries. If a mistake was made during the training database generation process then that particular entry can be overwritten at a later stage. If a similar mistake were made with the previous system's software, then the training database generation process would typically have to be restarted.

A component was coded to provide a graphical user interface to set the desired network output. In terms of the number of pixels in the image, more pixels clearly produce a better quality image. However, by increasing the number of pixels, the system sensitivity of each pixel will decrease. In addition, readings obtained from real tomography systems contain noise. Consequently, if the noise level is higher than the sensitivity level, the reconstructed image results become vague [24]. The minimum resolution of the system therefore determines the size of these pixels [17]. It was decided that once again a 10 by 10 matrix of pixels would be used to display the neural network reconstruction results. The details of this component and its operation are provided in Appendix K on page 150.

The high data capture rate of the sampling unit also reduces the time taken to generate a training or test database. It is often the case that more than one frame is to be stored for a particular configuration of air or gravel bubbles. Since all these training data points have the same desired output but whose inputs differ slightly due to electrical noise, the noise rejection performance of the reconstruction neural network is improved. Consequently, 25 frames were typically captured for every training or test configuration.

The following is a brief description of how one would use the software in order to generate a training or test database. It is not an attempt to explain the underlying implementation of the software, which can be determined by reading the commented program code in Appendix K on page 142. When the program opens, it automatically loads the most recently modified database. Since a new database is desired, selecting 'Close database' in the file menu closes this old database. A new database is created by selecting 'New database' and the user is prompted to provide a name for this new database. The software automatically configures the tables in this new database. As mentioned previously, it is typical to capture a number of versions of the same bubble configuration in order to improve the noise performance of the trained neural networks. 'Set the number of frames' under the edit menu

allows one to set the number of versions to capture. Having configured the software, one can now progress to capturing a database of readings.

'Add to database' in the edit menu opens a new window that is used for the capturing of database data. Figure 5.19 is a screen capture of this form and a full-scale version of this screen capture can be found in Appendix M on page 202. Firstly, it is necessary to set the source of the data, namely the top or bottom tomography system. The right arrow is clicked to create space in the database for a new set of samples. The desired network output is then drawn on the tomography component in the top left of the form. Simply by clicking on a pixel the phase of that particular pixel changes and, depending on which mouse button was pressed, to either seawater, gravel or air. The bar chart simply displays the corresponding volume fractions for each of the three phases. Having set the desired network output in software, it is necessary to replicate this bubble configuration in the actual rig. Once complete, 'Start sampling...' is clicked and the software will capture a set of frames corresponding to this bubble configuration, where the number of frames captured equals the desired number of versions. To add another entry to the database, the right arrow is once again clicked and the process is repeated. The left and right arrows allow one to navigate through the database, and if the end of the database is reached, creates space for another set of samples. The same process is followed for the generation of a test database.

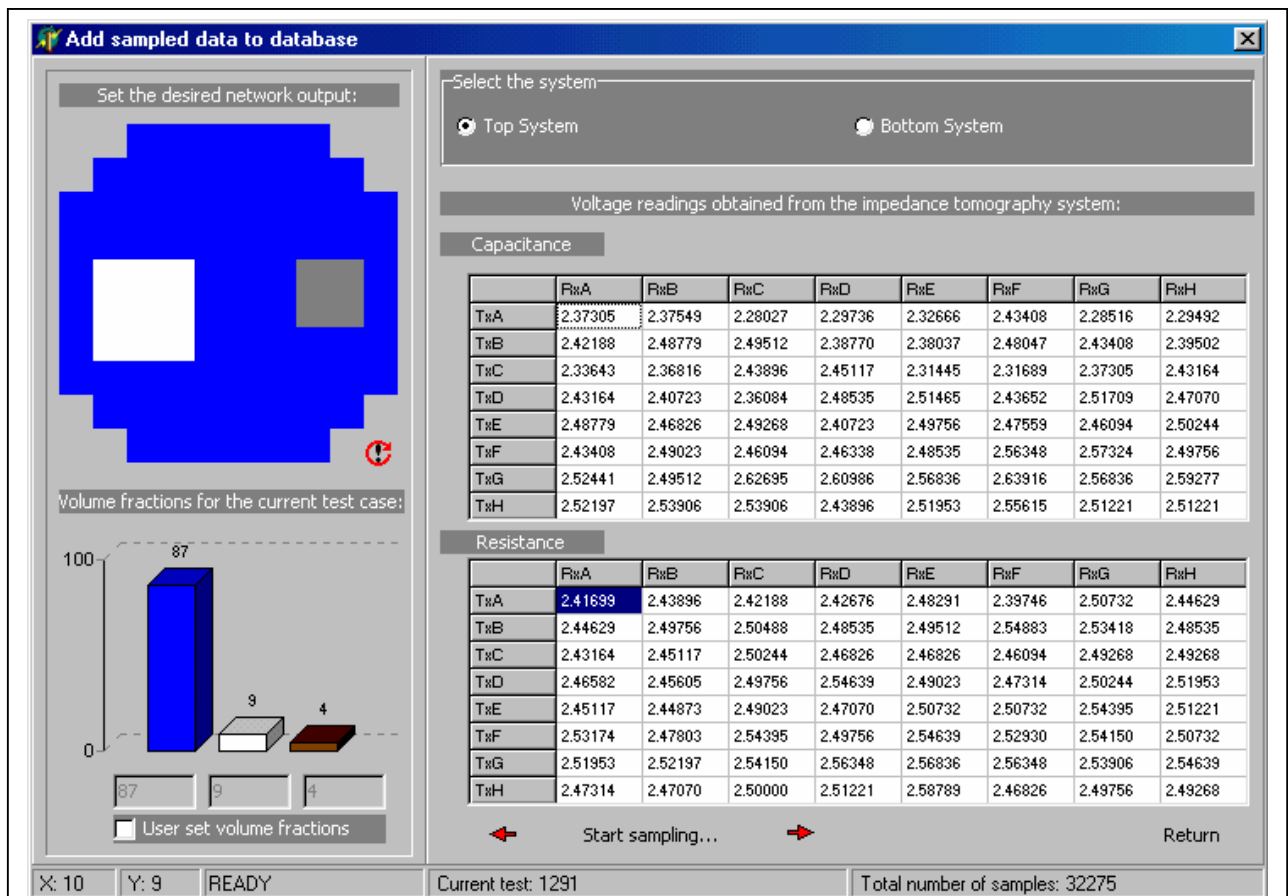


FIGURE 5.19
SCREEN CAPTURE OF THE DATABASE GENERATION FORM. AT THE TOP LEFT IS THE GRAPHICAL COMPONENT USED TO SPECIFY THE DESIRED NETWORK OUTPUT, WHERE BLUE REPRESENTS SEAWATER, WHITE REPRESENTS AIR AND BROWN REPRESENTS GRAVEL. THIS WAS A 0.09 VOLUME FRACTION AIR BUBBLE SITUATED NEAR THE LEFT-HAND EDGE OF THE RIG AND A 0.04 VOLUME FRACTION GRAVEL BUBBLE SITUATED NEAR THE RIGHT-HAND EDGE OF THE RIG. BELOW THIS COMPONENT IS A BAR GRAPH OF THE DESIRED VOLUME FRACTIONS. TO THE RIGHT OF THESE COMPONENTS ARE TWO TABLES OF THE VOLTAGES CORRESPONDING TO THE CAPACITANCE AND RESISTANCE READINGS RESPECTIVELY. BELOW THESE TABLES ARE THE TWO ARROWS USED TO NAVIGATE THROUGH THE DATABASE.

The data capture program provides an additional feature for the monitoring of the system electronics. Since 128 voltage readings are available, it is impractical to monitor the performance of the system in terms of individual voltages. By selecting 'Current samples' under the view menu, a form opens showing all 128 voltages in a tabular format that is continuously updated. A screen capture of this form can be found in Appendix M on page 203. A further feature is provided whereby frames of data can be saved to a single text file at a specified rate. This is used for the assessment of the system drift, since this text file can be loaded in a spreadsheet and the voltages plotted. Having generated a training and test database, the neural network program is used to train a neural network to perform reconstruction, and this will be examined in the following section.

5.3.3 Neural network reconstruction software description

As before, the underlying structure of the neural network program is very similar to the software developed as part of the author's previous research. The commented program code is provided in Appendix L on page 152. The following is not an attempt to analyse the implementation of the neural networks. Instead it describes briefly how one would train and verify a neural network.

Firstly, the training and test databases must be loaded. Selecting 'Load database' in the file menu opens a form where the specific details of the database are entered. A screen capture of this form is included in Appendix N on page 204. In particular, the user is required to specify whether image reconstruction or volume fraction prediction is required, the names of the training and test databases and the number of versions and range of training and test data points to load. Once this information has been specified, the required data will be loaded by clicking 'Load databases...'. If 'Calibrate' is selected then additional calibration data will also be required. This will be discussed at a later stage once the operation of the software calibration feature has been explained. After the data has been loaded, it must be pre-processed. Standardisation of the training data is performed when 'Pre-process data...' is clicked.

On returning to the main menu, the user has the option of loading a previous set of network weights, thus allowing one to perform tests without having to retrain a neural network. Since this is a discussion on training and testing a new neural network, the next stage requires the setting of the neural network parameters. Each network trained has a different test number, which can be specified by selecting 'Set test number' in the network training menu. All network training results, network weights and other data are saved according to this test number. Therefore, if one wished to load the network weights for a specific network, then it would be necessary to specify the corresponding test number beforehand.

Set the network training parameters

General parameters

Maximum number of epochs: 2000

Number of hidden layer neurons: 5

☒ Perform early stopping to prevent over-fitting

☒ Conduct model search

Early stopping parameters

Number of failures before training stops: 2

Stopping condition:

☒ Sum total of volume fraction errors

Network specific parameters

☐ Use gradient descent ☒ Use Resilient back-propagation

Specify the initial delta value: 0.0001

Specify the maximum delta value: 50

Specify the performance refresh rate: 1

☒ View the training database performance measures

Model search parameters

Minimum number of hidden layer nodes: 5

Maximum number of hidden layer nodes: 100

Increment: 5

Return

FIGURE 5.20
SCREEN CAPTURE OF THE NEURAL NETWORK PARAMETER FORM. THIS SHOWS A TYPICAL CONFIGURATION FOR A DOUBLE-LAYER FEED-FORWARD NEURAL NETWORK TRAINED USING RESILIENT BACK-PROPAGATION TO PERFORM A VOLUME FRACTION PREDICTION. A MODEL SEARCH WILL BE CONDUCTED TO DETERMINE THE OPTIMAL NUMBER OF HIDDEN LAYER NEURONS. THE MINIMUM NUMBER OF HIDDEN LAYER NEURONS IS SET AT 5 AND THE MAXIMUM NUMBER OF HIDDEN LAYER NEURONS IS SET AT 100. EARLY STOPPING IS ALSO PERFORMED TO PREVENT OVER-FITTING.

'Set the network training parameters' opens a form where the desired training parameters are specified. These include the type of training algorithm to use, whether early stopping should be performed, the number of hidden layer neurons for a double-layer neural network and so on. A model search is simply an iterative search to determine the optimal number of hidden layer neurons in a double-layer feed-forward neural network and is a fairly time consuming process. The other training parameters are fairly self-explanatory and figure 5.20 is a screen capture of this form. A full-scale version of this screen capture is included in Appendix N on page 205. This shows a typical configuration of a double-layer feed-forward neural network trained using Resilient back-propagation to perform a volume fraction prediction.

Before network training commences, the network weights must be randomly initialised and this is performed by selecting 'Randomly initialise network weights'. 'Start training' opens a form that allows the user to monitor the network training process, and in particular, the generalisation performance of the neural network on the test database. A screen capture of this form can be found in Appendix N on page 206. Network training can be stopped temporarily by clicking 'Stop training...' if it is necessary to perform some other task on the computer during network training. The training will resume on clicking 'Start training...'. Having trained a neural network, the network weights, performance results and pre-processing data need to be saved in order for the network to be reused at a later date. 'Save network weights', 'Save network performance' and 'Save network pre-processing' in the file menu perform these functions. The network performance can then be tested using the various options provided in the network testing menu.

'Validate database' simply displays the neural network reconstruction results for each of the test data points sequentially. It also displays the desired network output so that a comparison can be made. A screen capture of this form is included in Appendix N on page 207. 'Validate current samples' is an extension of this where, instead of using data from the test database, voltage readings are continuously captured from the impedance tomography rig and the reconstructions performed and displayed. In this situation, there is no desired output. Instead a comparison is made between the network prediction and the bubble configuration within the rig at that point in time. The desired frame-rate can be set, although the maximum achieved is typically only 20frames/s due to the simplicity of the software implementation. If the maximum frame-rate of 280frames/s is required, then the 'Real-time sampling' form must be used. The operation of this unit will be examined in greater detail when the dynamic performance of the FDM impedance tomography system is analysed in Chapter 7.

CHAPTER 6

STATIC SITUATION RESULTS FOR THE 16-ELECTRODE IMPEDANCE TOMOGRAPHY SYSTEM

The following chapter examines the performance of the 16-electrode FDM impedance tomography system for static bubble configurations. Further, these tests are performed with only one of the measurement sections operating at any one time. Performance assessments are generally made with respect to a reconstruction algorithm due to the large number of measurement variables in a tomography system. Although the desired output is the volume fraction of each of the three phases and their corresponding component velocities, image reconstructions will also be performed as a means of demonstrating the wide range of applications to which such a system is applicable. The following sections will examine specifically the configuration and assessment of the top measurement section. It should be noted that the same process was followed for the bottom measurement section; due to the similarity of the results achieved, this will not be discussed in detail.

Before training database generation could commence, it was necessary to tune the system for optimal performance. Firstly, a blanking-off disk was constructed to seal the top measurement section. The flange clamps used to assemble the laboratory-scale rig were used to secure the blanking-off disk to the base of the measurement section with a rubber insertion gasket. The measurement section was then filled with seawater and the system was switched on. The sample controller board was connected to the PC30G card and the sampling software running on the reconstruction computer provided a means of monitoring all 128 voltages. In this way, it was possible to observe trends in the voltages as bubbles were introduced.

It was noted that the introduction of a bubble resulted in the increase of certain readings. However, for both the capacitance and conductance measurements it is expected that the readings should decrease, since air has both a lower permittivity and conductivity than seawater. This phenomenon has been explained for the case of multiple transmitters operating in parallel for conductance tomography by Hervieu and Seleglim as follows [42]: the presence of a non-conducting object in the region of an electrode results in a current deficiency at that particular electrode and a current excess at its neighbours. For larger non-conductive objects, this current deficiency extends to multiple electrodes and although the current excess is still present, it is compensated by an overall current reduction resulting from the global reduction in conductivity. For the case of capacitance tomography, Xie *et al* explain this situation as follows [47]: the introduction of a different phase into the vessel will redistribute the electric flux lines within the vessel. Consequently, certain detector electrodes may absorb more electric flux lines than before, resulting in over-shooting effects. Subsequently, it became clear that the voltage levels in the system could not be tuned with only seawater in the rig as the maximum expected capacitance and conductance readings, since the introduction of a bubble could result in the saturation of certain op-amp outputs.

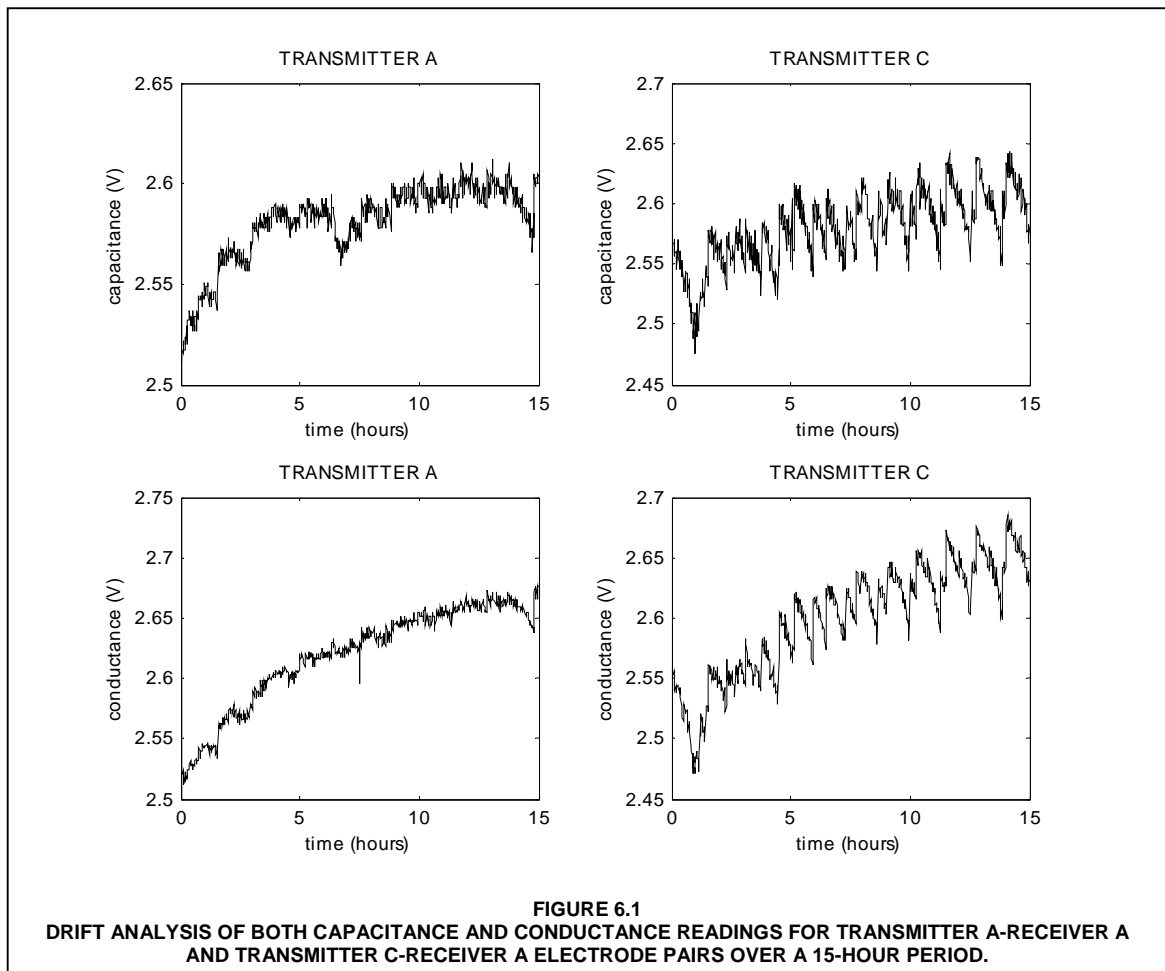
Saturation of the op-amp outputs would introduce a further non-linearity into the readings making the reconstruction task even more complex. To ensure that such a saturation would not occur, the amplifications of all op-amp stages within the system were reduced so as to provide a safety margin. As an example, the gains of the LMF100 switched capacitor filters were reduced so that the DC output voltage levels were at approximately 2.5V, even though the LMF100 switched capacitor filters were on a 5V split-supply and could produce output signals greater than 3.7V. The output capacitance and conductance voltages were all tuned to be approximately 2.5V with the measurement section full of seawater. As mentioned previously, these voltages included a ripple component

resulting from the harmonic mixing in the multiplication stage of the synchronous detectors. The average magnitude of this ripple was 21mV peak-to-peak for the capacitance readings and 14mV peak-to-peak for the conductance readings. Having tuned the FDM impedance tomography system, it was necessary to analyse the long-term stability of these readings.

6.1 Analysis of Capacitance and Conductance Measurement Drift

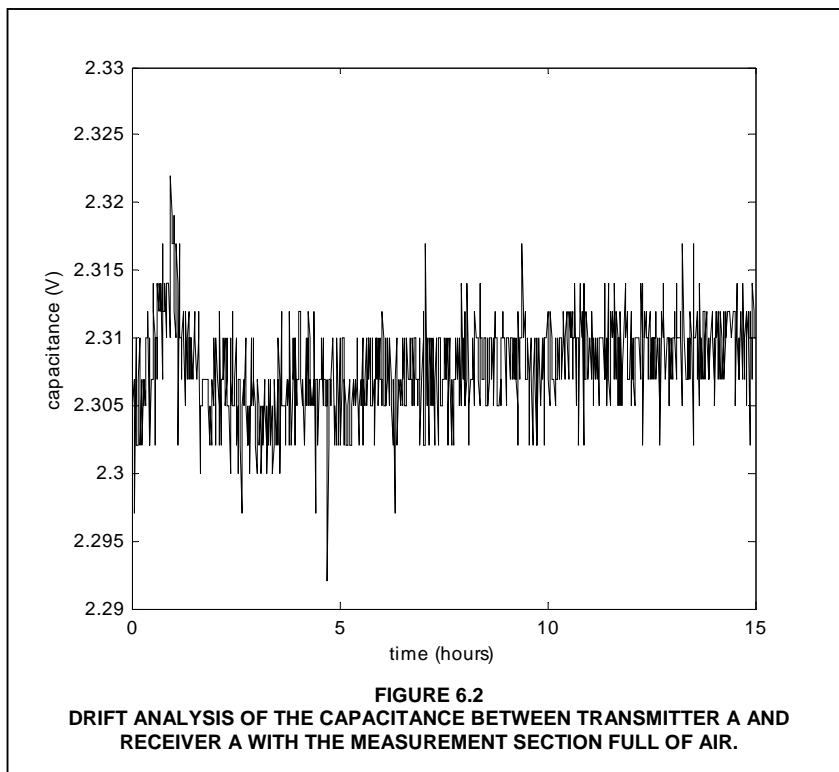
To analyse the long-term stability of the capacitance and conductance voltage readings, the sampling software was modified to include a data-logging feature. This feature saves frames of data at a set rate to a comma-separated text file, which is then loaded into a spreadsheet for analysis. Since only a drift analysis is required, a logging rate of one frame every minute should give an indication of how the readings vary over a 24-hour period.

A drift analysis of the FDM impedance tomography system revealed considerable drift of both the capacitance and conductance readings. Figure 6.1 illustrates the drift of both capacitance and conductance voltages over a 15-hour period. Tests revealed that the behaviour of the drift was independent of the particular receiver analysed. As is evident, the drift of the capacitance between receiver A and transmitter A is very similar to the drift of the conductance between receiver A and transmitter A. This also extends to completely different receivers. For example, the drift of the capacitance between receiver A and transmitter A is very similar to the drift of the capacitance between receiver B and transmitter A. From this it was concluded that the drift was related to the transmitter outputs and not the receivers. The average capacitance voltage drift was 121mV and the average conductance voltage drift was 167mV over a 15-hour period.



Standard conductance measurement systems use the four-electrode adjacent-pair measurement protocol to ensure that the readings are independent of any electrochemical reactions taking place within the vessel. This was discussed in detail in Chapter 2. However, the current FDM impedance tomography technique is limited to two-electrode conductance measurements and is therefore sensitive to polarisation voltages, as well as variations in the contact impedance between the conductance probe and the seawater. Specifically, corrosion results in the fouling of the conductance probe. Since this contact impedance is included in the conductance measurement, an increase in contact impedance results in a decrease in the measured conductance. Further, DC electrochemical potentials exist at the electrodes and these voltages differ for each electrode [37]. Research is currently being done on the design of a FDM impedance tomography system to perform four-electrode adjacent-pair conductance measurements.

During a drift analysis, it was observed that bubbles would form on the conductance probes. The formation of these bubbles explains the almost oscillatory behaviour of the readings for transmitter C in figure 6.1 on page 67. As the bubble is formed so the contact impedance between the seawater and conductance probe increases as a result of the reduced contact area. Consequently, the conductance reading decreases steadily. As soon as the bubble is large enough and breaks away, the contact impedance drops and subsequently, the conductance reading increases sharply. Further, it was observed that the conductance probes would corrode over time. Consequently, it was decided that the drift of the capacitance and conductance readings was the result of the electrochemical reactions taking place within the rig and was not related to electronic component drift.



To prove this assumption correct, a drift analysis was performed with the measurement section empty. Subsequently, no electrochemical reactions could take place and only electronic component drift would be monitored. The resulting drift analysis is plotted in figure 6.2, which is the drift of the capacitance reading between transmitter A and receiver A. As is evident from figure 6.2, minimal drift occurs when the measurement section does not contain seawater. The remaining drift is probably the result of temperature variations. From this result it was concluded that the capacitance and conductance voltage drift was the result of

electrochemical reactions taking place between the conductance probes and the seawater.

In order to minimise this drift, various alternate materials were tested for the conductance probes. Initially, the conductance probes were 5mm stainless steel machine screws. These were replaced with type-316 marine-grade stainless steel. In addition, carbon probes were tested since carbon is considered both durable and chemically resistant. To provide a reference, the measurement section was filled with potable water and a drift analysis was performed. Since potable water has a far lower conductance than seawater, the drift resulting from electrochemical reactions should be greatly reduced. Table 6.1 compares the average drift of both the capacitance and conductance readings for the different conductance probe materials tested over a 15-hour period. It also includes the drift measured when the measurement section is filled with potable water.

TABLE 6.1 COMPARISON BETWEEN THE DRIFT OF THE CAPACITANCE AND CONDUCTANCE READINGS OVER A 15-HOUR PERIOD USING DIFFERENT CONDUCTANCE PROBE MATERIALS.				
	STANDARD STAINLESS STEEL	MARINE-GRADE STAINLESS STEEL	CARBON PROBE	POTABLE WATER REFERENCE
CAPACITANCE	121mV	235mV	159mV	28mV
CONDUCTANCE	167mV	269mV	264mV	34mV

Since insufficient funds were available to investigate more advanced conductance probe materials, it was decided that all future tests should be done with potable water. This required an increase of the conductance receiver gain. A larger input resistor was used to increase the conductance receiver output and specifically an 820 Ω resistor was used, whereas a 15 Ω resistor had been used for seawater tests. This is only a temporary solution and clearly more research must be done on this topic before an industrial prototype is developed. Further, it is believed that in the intended application the constant material flow through the measurement section would assist in keeping the conductance probes clear of bubbles and clear away corrosion deposits. Subsequent work by the author's colleague, Mr A Giannopoulos, in which the water is circulated in a static rig by means of a pump, has shown this assumption to be correct [62].

6.2 Training and Test Database Generation

Training a neural network requires the generation of a representative training database. As mentioned previously, a bubble placement system, developed as part of the author's previous research, provided a convenient way to systematically vary the contents of the vessel. The bubble placement system consists of a 'bed-of-nails' placed at the base of the vessel and a clear acrylic lid. Whereas in previous work, long cylindrical bubbles were used that extended the full length of the vessel, it was decided that shorter bubbles with rounded ends would be more realistic. Consequently, it was necessary to change the concept of a 'bed-of-nails' into a 'bed-of-hooks' so that the air bubbles could be anchored at specific positions within the measurement section, remembering that an 'air bubble' is actually a contiguous polystyrene foam mass. The blanking-off disk mentioned previously was used to provide this array of anchor points corresponding to specific positions within the vessel. Once again, a clear acrylic lid was constructed with the matrix of 10 by 10 pixels clearly marked. Sequences of holes were drilled in this lid and the gravel bubbles were suspended from hooks. In this way, it is possible to simulate various static configurations of both air and gravel masses. Figure 6.3 on page 70 is a photograph of the bubble placement system. Each pixel on the clear acrylic lid is numbered and this numbering corresponds to the graphical user interface of the data capture software used to specify the desired network output.

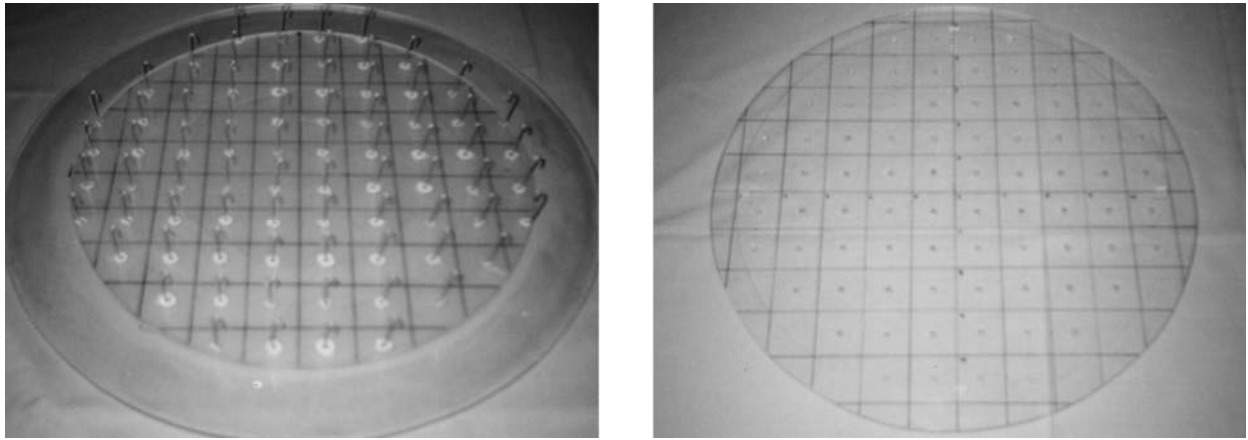
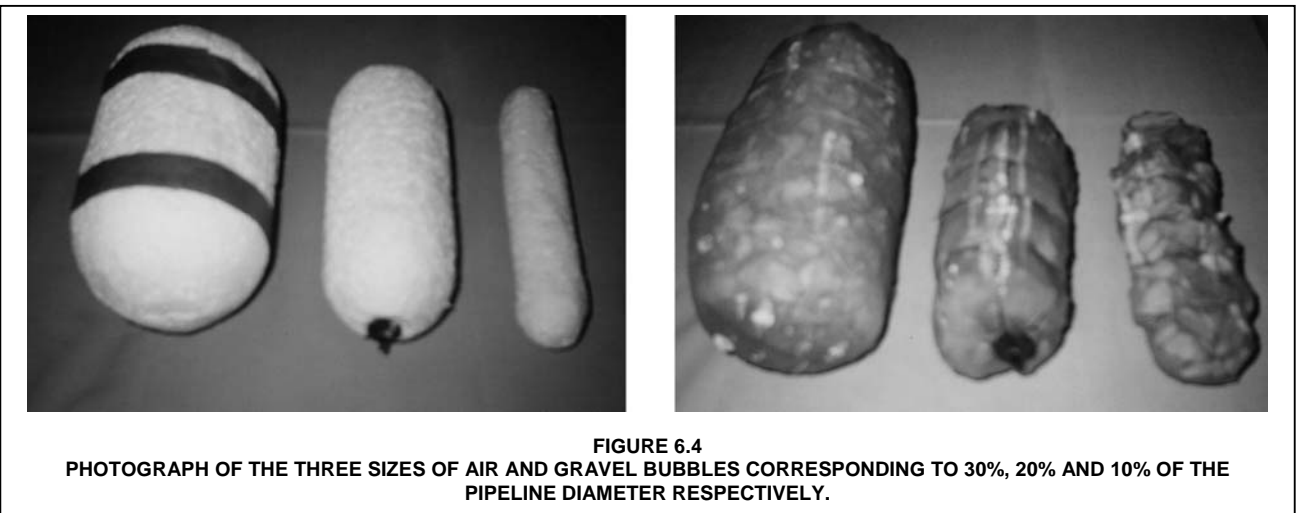


FIGURE 6.3
PHOTOGRAPH OF THE 'BED-OF-HOOKS' BASE PLATE AND CLEAR ACRYLIC LID USED AS THE BUBBLE PLACEMENT SYSTEM FOR THE GENERATION OF STATIC TESTS.

Various changes were implemented in the generation of the training and test database. Firstly, the bubbles in both the training and test databases were not allowed to come into contact with the conductance probes. When a bubble comes into contact with a conductance probe, the measurements change dramatically. In contrast, the changes introduced by moving a bubble at the centre of the pipeline are much smaller. By not allowing the bubbles to come into contact with the conductance probes, the measurement range is significantly reduced. Subsequently, it is easier to train a neural network to detect the minor changes resulting from the movement of bubbles at the centre of the pipeline. In contrast, the previous neural networks were dominated by the bubbles at the edge of the pipeline and consequently the resolution at the centre was poor. Tests were conducted to compare the performances of neural networks trained on data that included bubbles at the edges and data that did not. Results showed that a greater resolution and accuracy was consistently achieved for the latter. A drawback of this approach is that erroneous results are achieved when a bubble is allowed to come into contact with a conductance probe. An example of this situation will be analysed in Chapter 7.

Secondly, the training database was extended to include combinations of air bubbles, combinations of gravel bubbles and combinations of both air and gravel bubbles. Previously, the training database had consisted only of single air bubbles and single gravel bubbles. In this situation, the reconstruction of more complex configurations is dependent on the generalisation capability of the neural network. Further, it was shown that the neural network reconstruction algorithms consistently produced incorrect reconstructions for situations that contained the masking effect. An example of the masking effect is the configuration consisting of a bubble near the edge and a bubble at the centre. In this configuration, the neural network was unable to detect the bubble at the centre and only detected the bubble near the edge since the reconstruction is dominated by the changes introduced by the bubble near the edge. Although not verified at the time, it was believed that the neural network would perform better reconstructions of these situations if the training database included examples of this masking effect. Consequently, it was decided that the training database for the 16-electrode FDM impedance tomography system should be extended to include these configurations, as well as combinations of both air and gravel bubbles. Tests were conducted to compare the performances of neural networks trained on the extended training databases and those trained on the original databases. Results showed that the neural networks trained on the extended databases consistently outperformed those trained on the original databases in terms of the detection of bubbles at the centre of the pipeline and the accurate reconstruction of configurations exhibiting the masking effect.

Finally, smaller bubbles were used in the generation of the training database in an attempt to extract the maximum possible resolution from the 16-electrode FDM impedance tomography system. Previously, the smallest bubble used in the training database generation corresponded to 20% of the pipeline diameter, thus setting the maximum achievable resolution at 20%. This bubble size corresponds to four pixels in the 10 by 10 image reconstruction. In tests, the previous system was only able to detect the 20% bubble towards the edges of the pipeline. For the 16-electrode FDM impedance tomography system, the smallest bubble simulated corresponds to 10% of the pipeline diameter and is represented by a single pixel in the 10 by 10 image reconstruction. In total, the bubble sizes simulated correspond to 10%, 20% and 30% of the pipeline diameter and represent volume fractions of 0.01, 0.04 and 0.09 when situated within the measurement volume. In terms of the lengths of the bubbles, all bubbles simulated are 190mm long thus ensuring the bubble extends beyond the sensitivity region of the measurement electrodes. Figure 6.4 is a photograph of the simulated air and gravel bubbles used in the training and test database generation. The diameter of the 10% bubble is 35mm.



Although the use of potable water dramatically reduced the drift of the capacitance and conductance readings, it was observed that the readings would still drift. It is believed that this drift is the result of temperature variations. Since the task of generating the training and test database takes place over a couple of days, any drift of the readings will have a detrimental effect on the networks trained using these databases. In addition, it was observed that the properties of the water would vary as a result of the impurities introduced by the gravel bubbles. Subsequently, the capacitance and conductance readings would drift during database generation. It was therefore decided that some form of software calibration feature should be provided to compensate for this remaining drift.

The software calibration feature calculates the changes resulting from the introduction of a bubble. At the start of the training database, a frame of voltages is captured with the measurement section filled only with water. This will provide a starting reference. All training data points are then captured normally as before. On completion of the training database generation, a further frame of voltages is once again captured with the measurement section filled only with water thus providing an end reference. This process is repeated for the generation of a test database.

When the data is loaded into the neural network reconstruction software, the following correction is applied to each voltage using the start and stop points as references

$$V_{jk} = V_{jk} - \left(V_{1k} + \frac{j}{N} (V_{Nk} - V_{1k}) \right) \quad (6.1)$$

where k is the index of a particular voltage in the 128 voltages corresponding to a complete frame, j is the index of this particular frame in the database, N is the total number of frames in the database and V_{1k} and V_{Nk} represent the water only tests conducted at the beginning and the end of the database generation respectively. In this way, the neural network reconstructions are based on the differences resulting from the introduction of a bubble instead of the absolute voltage readings. Subsequently, any minor drift during database generation is corrected for. Before an on-line real-time test is performed, a frame of voltages is captured with the measurement section filled only with water. These reference voltages are then subtracted from the real-time captured voltages during testing. The advantage of this approach is that the neural networks trained are unaffected by changes to the properties of the water in the rig and it is not necessary to retune the synchronous detector boards each time the water in the rig is replaced. Clearly, this will only work to a point. If the electrical properties have changed to such a degree that saturation is occurring in the measurement electronics, then retuning would be necessary.

Using the techniques discussed above, a training and test database were generated for both the top and bottom measurement sections. These databases were of static configurations and consisted of both air and gravel bubbles, as well as different combinations of air and gravel bubbles. The author's undergraduate thesis compared the performances of the different neural networks for this reconstruction task and a paper detailing the major results of this research is included in Appendix O on page 211. Since the emphasis of this thesis is more on the dynamic performance of an FDM impedance tomography system, the following section will simply tabulate the reconstruction results achieved and provide examples of results for the top measurement section.

6.3 Neural Network Reconstruction Results

Only limited accuracy had been achieved previously with three-phase reconstructions. It was therefore decided that a better assessment of the 16-electrode FDM impedance tomography system performance could be achieved using the simpler situation of a two-phase air-water reconstruction.

6.3.1 Two-phase air-water reconstruction results

Multiple single-layer feed-forward neural networks were trained to perform the two-phase air-water image reconstruction and the averaged results are detailed in table 6.2, which includes a comparison to the results for the 8-electrode prototype. As is evident in table 6.2, the 16-electrode system outperforms the 8-electrode system for all performance assessment criteria considered. This was expected since more capacitance and conductance readings are obtained from the 16-electrode system than the 8-electrode system and therefore an increase in resolution is achieved.

TABLE 6.2 COMPARISON BETWEEN THE PERFORMANCE OF THE 16-ELECTRODE FREQUENCY DIVISION MULTIPLEXED IMPEDANCE TOMOGRAPHY SYSTEM AND THE 8-ELECTRODE PROTOTYPE FREQUENCY DIVISION MULTIPLEXED IMPEDANCE TOMOGRAPHY SYSTEM FOR A TWO-PHASE AIR-WATER IMAGE RECONSTRUCTION USING A SINGLE-LAYER FEED-FORWARD NEURAL NETWORK. ALL RESULTS REPRESENT THE ABSOLUTE ERROR AND ARE ROUNDED OFF TO TWO SIGNIFICANT DIGITS.			
PERFORMANCE MEASURE	16-ELECTRODE LABORATORY-SCALE FDM IMPEDANCE TOMOGRAPHY SYSTEM		8-ELECTRODE PROTOTYPE FDM IMPEDANCE TOMOGRAPHY SYSTEM
	TOP SYSTEM	BOTTOM SYSTEM	
THRESHOLD ERROR (%)	2.1	2.2	9.1
AIR VOLUME FRACTION ERROR (%)	1.4	1.4	5.0
WATER VOLUME FRACTION ERROR (%)	1.4	1.4	5.0

Figure 6.5 on page 74 is a sequence of screen captures illustrating the neural network reconstruction results for certain test cases from the top measurement system. For each screen capture, the desired output is the right-hand frame and the network prediction is the left-hand frame. Certain conclusions can be made regarding the 16-electrode FDM impedance tomography system based on the screen captures in figure 6.5:

- By extending the training database to include examples of the masking effect, the neural network reconstruction accuracy for these configurations is greatly improved. For example, the test cases in figure 6.5 reveal that when a bubble is situated towards the edge and another bubble is situated at the centre of the pipeline, the bubble at the centre is also detected. Previously, only the bubble towards the edge would have been detected.
- The resolution of the system has improved to where the 10% bubble can be detected at the centre of the pipeline. Previously, the 8-electrode prototype was unable to detect the 20% bubble at the centre of the pipeline. By increasing the number of electrodes to 16 and by not allowing the training database bubbles to come into contact with the conductance probes, the resolution at the centre of the pipeline has increased to 10% of the pipeline diameter, corresponding to a 0.01 volume fraction. Since the 10% bubble is the smallest bubble with which the network has been trained, this represents the maximum achievable resolution for the current system. However, it is possible that greater resolution could be achieved through training the network with smaller bubbles.
- By using the 'Test current samples' feature of the reconstruction program, it was determined that these results could be replicated for on-line reconstruction results, where the readings are captured directly from the system and a reconstruction is performed. Further, the reconstruction results were unaffected by minor variations in the properties of the water thus proving the correct operation of the calibration feature.

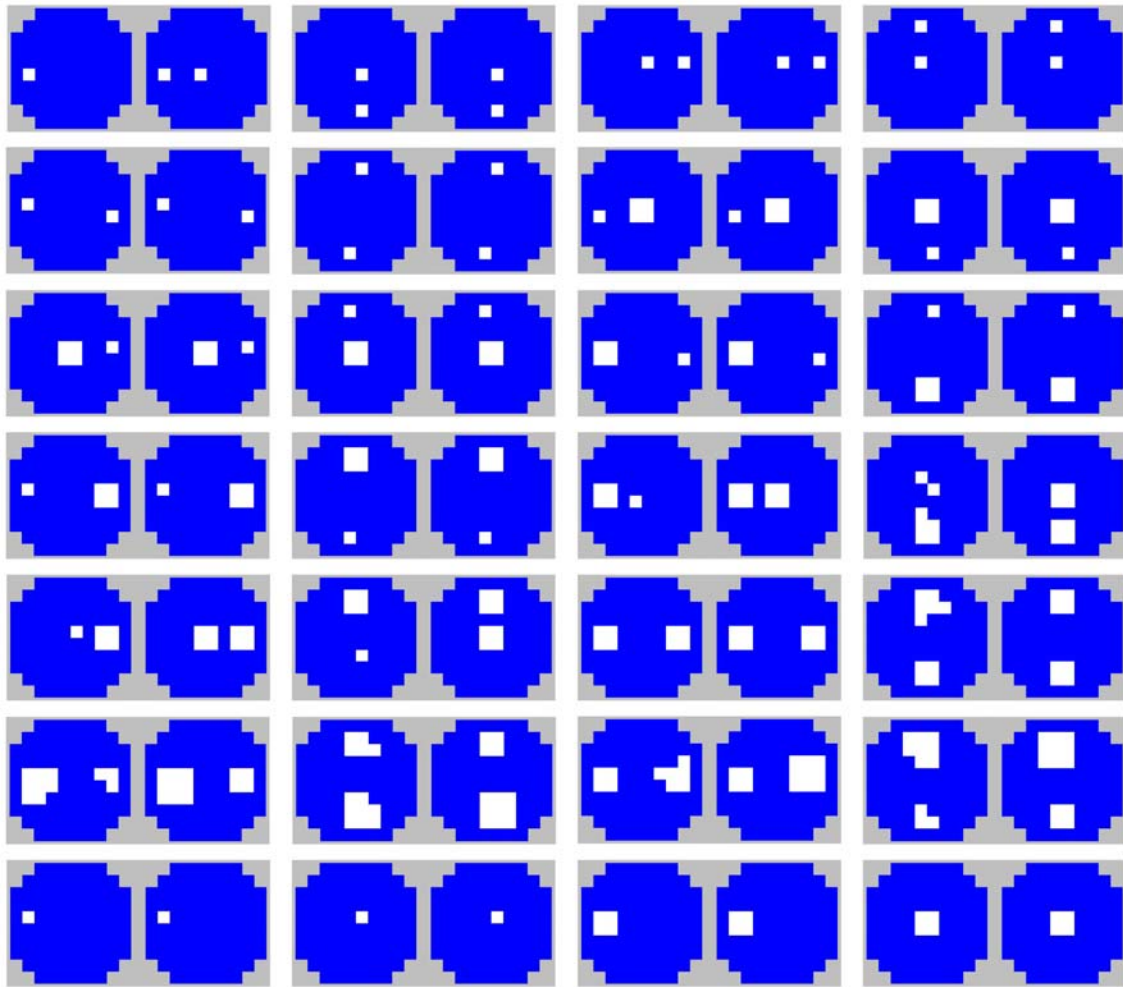


FIGURE 6.5
SCREEN CAPTURES OF TEST CASES FOR THE TWO-PHASE AIR-WATER IMAGE RECONSTRUCTION OF THE TOP FREQUENCY DIVISION MULTIPLEXED IMPEDANCE TOMOGRAPHY SYSTEM, WHERE THE DESIRED OUTPUTS ARE ON THE RIGHT AND THE NETWORK PREDICTIONS ARE ON THE LEFT IN EACH SCREEN CAPTURE. IN EACH FRAME, THE BLACK IS THE WATER PHASE, THE WHITE IS THE AIR PHASE AND THE LIGHT GREY REPRESENTS THE AREA OUTSIDE THE PIPELINE.

Previous research has shown that greater volume fraction prediction accuracy can be achieved by training a neural network to predict the volume fractions directly [3, 4]. In addition, the results are obtained faster than if an image reconstruction is first performed. Multiple single-layer feed-forward neural networks were trained to perform the two-phase air-water volume fraction prediction and the averaged results are detailed in table 6.3. As is evident from table 6.3, an improvement in the accuracy of the volume fraction prediction was achieved for both the top and bottom system compared to the image reconstruction results considered in table 6.2. The following section examines the reconstruction results for static three-phase air-gravel-water configurations.

TABLE 6.3
PERFORMANCE OF THE 16-ELECTRODE FREQUENCY DIVISION MULTIPLEXED IMPEDANCE TOMOGRAPHY SYSTEM FOR A TWO-PHASE AIR-WATER VOLUME FRACTION PREDICTION USING A SINGLE-LAYER FEED-FORWARD NEURAL NETWORK. ALL RESULTS REPRESENT THE ABSOLUTE ERROR AND ARE ROUNDED OFF TO TWO SIGNIFICANT DIGITS.

PERFORMANCE MEASURE	16-ELECTRODE LABORATORY-SCALE FDM IMPEDANCE TOMOGRAPHY SYSTEM	
	TOP SYSTEM	BOTTOM SYSTEM
AIR VOLUME FRACTION ERROR (%)	0.20	0.18
WATER VOLUME FRACTION ERROR (%)	0.20	0.18

6.3.2 Three-phase air-gravel-water reconstruction results

Multiple single-layer feed-forward neural networks were trained to perform the three-phase air-gravel-water image reconstruction and the averaged results are detailed in table 6.4, which includes a comparison to the 8-electrode prototype system. The networks employ a 1-of-C output encoding and early stopping is used to prevent over-fitting. As is evident from table 6.4, the 16-electrode FDM impedance tomography system outperforms the 8-electrode prototype on all the performance assessment criteria considered. This is consistent with the results of the two-phase air-water image reconstruction.

TABLE 6.4 COMPARISON BETWEEN THE PERFORMANCE OF THE 16-ELECTRODE FREQUENCY DIVISION MULTIPLEXED IMPEDANCE TOMOGRAPHY SYSTEM AND THE 8-ELECTRODE PROTOTYPE FREQUENCY DIVISION MULTIPLEXED IMPEDANCE TOMOGRAPHY SYSTEM FOR A THREE-PHASE AIR-GRAVEL-WATER IMAGE RECONSTRUCTION USING A SINGLE-LAYER FEED-FORWARD NEURAL NETWORK. ALL RESULTS REPRESENT THE ABSOLUTE ERROR AND ARE ROUNDED OFF TO TWO SIGNIFICANT DIGITS.			
PERFORMANCE MEASURE	16-ELECTRODE LABORATORY-SCALE FDM IMPEDANCE TOMOGRAPHY SYSTEM		8-ELECTRODE PROTOTYPE FDM IMPEDANCE TOMOGRAPHY SYSTEM
	TOP SYSTEM	BOTTOM SYSTEM	
THRESHOLD ERROR (%)	2.7	2.5	13
AIR VOLUME FRACTION ERROR (%)	1.2	1.2	2.9
GRAVEL VOLUME FRACTION ERROR (%)	1.5	1.6	5.3
WATER VOLUME FRACTION ERROR (%)	0.80	0.86	5.6
SUM VOLUME FRACTION ERROR	3.5	3.7	14

Figure 6.6 on page 76 includes screen captures of specific test cases from the top measurement system. For each screen capture, the desired output is the right-hand frame and the network prediction is the left-hand frame. These test cases confirm the conclusions of the two-phase air-water image reconstruction results. However, for the three-phase reconstruction, it is evident from figure 6.6 that the neural network is not always able to distinguish between the gravel and air phases. In certain test cases, the 0.04 volume fraction gravel bubble is occasionally detected as an air-gravel combination, or vice versa. Clearly, the separation between the gravel and air phases represents the most complex requirement of the reconstruction algorithm. As mentioned previously, the conductance of the gravel and air phases can be considered the same. Subsequently, any distinction between these two phases is based solely on their differences in dielectric constants. In addition, these results are for the ideal situation of static bubbles that are contiguous masses of gravel chips or polystyrene foam cylinders. It is not known how the neural network would perform for the situation of a moving distributed three-phase mixture. This issue will be discussed further at a later stage.

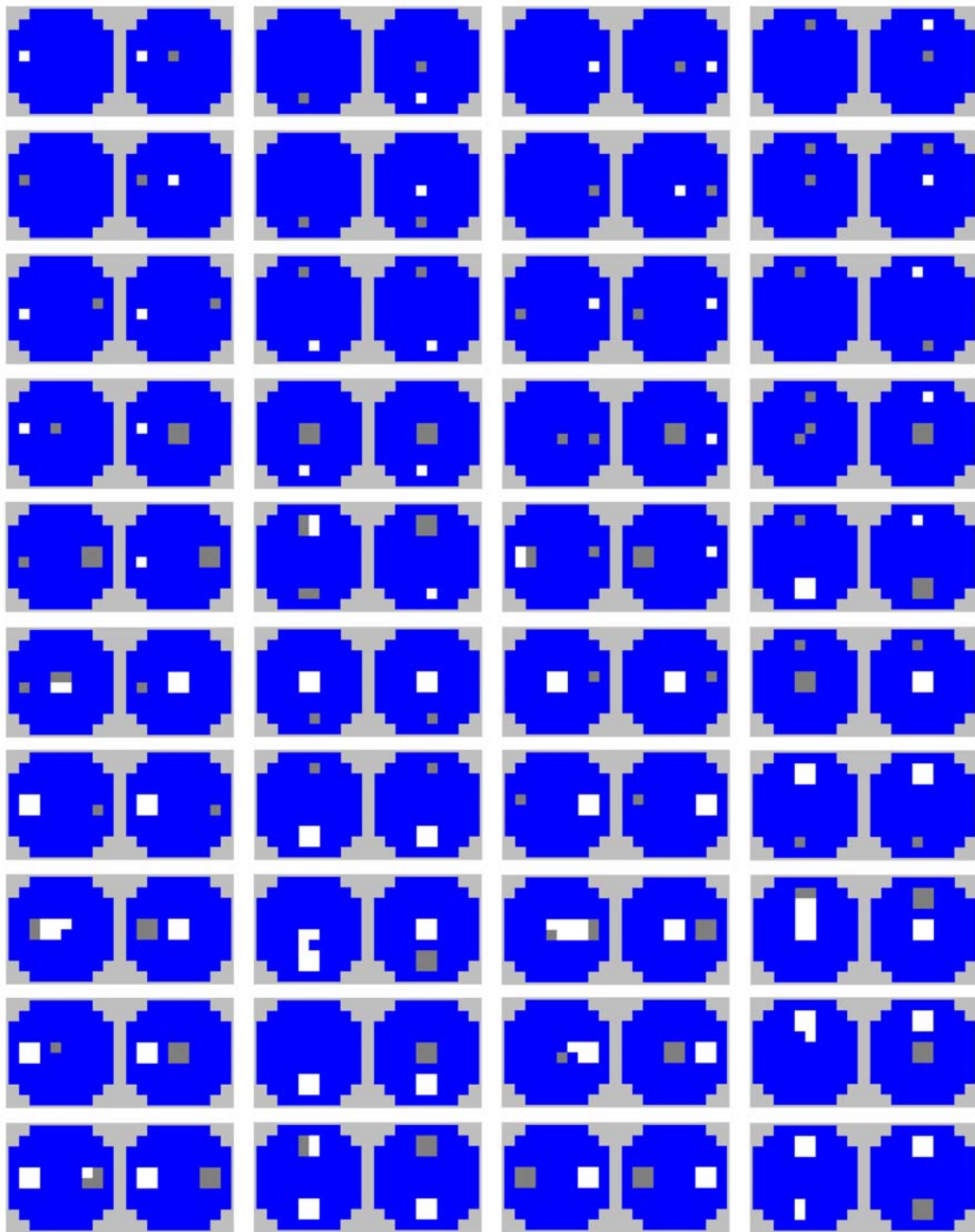


FIGURE 6.6
SCREEN CAPTURES OF TEST CASES FOR A THREE-PHASE AIR-GRAVEL-WATER IMAGE RECONSTRUCTION OF THE TOP FREQUENCY DIVISION MULTIPLEXED IMPEDANCE TOMOGRAPHY SYSTEM, WHERE THE DESIRED OUTPUTS ARE ON THE RIGHT AND THE NETWORK PREDICTIONS ARE ON THE LEFT OF EACH SCREEN CAPTURE. AS BEFORE, THE BLACK IS THE WATER PHASE, THE WHITE IS THE AIR PHASE, THE DARK GREY IS THE GRAVEL PHASE AND THE LIGHT GREY REPRESENTS THE AREA OUTSIDE THE PIPELINE.

Multiple single-layer feed-forward neural networks were trained to perform the three-phase air-gravel-water volume fraction prediction and the averaged results are detailed in table 6.5.

TABLE 6.5 PERFORMANCE OF THE 16-ELECTRODE FREQUENCY DIVISION MULTIPLEXED IMPEDANCE TOMOGRAPHY SYSTEM FOR A THREE-PHASE AIR-GRAVEL-WATER VOLUME FRACTION PREDICTION USING A SINGLE-LAYER FEED-FORWARD NEURAL NETWORK. ALL RESULTS REPRESENT THE ABSOLUTE ERROR AND ARE ROUNDED OFF TO TWO SIGNIFICANT DIGITS.		
PERFORMANCE MEASURE	16-ELECTRODE LABORATORY-SCALE FDM IMPEDANCE TOMOGRAPHY SYSTEM	
	TOP SYSTEM	BOTTOM SYSTEM
AIR VOLUME FRACTION ERROR (%)	1.5	1.5
GRAVEL VOLUME FRACTION ERROR (%)	2.1	2.2
WATER VOLUME FRACTION ERROR (%)	0.78	0.77
SUM VOLUME FRACTION ERROR	4.4	4.5

As is evident from table 6.5, no major improvement in the accuracy of the volume fraction predictions was obtained compared to the image reconstruction results examined in table 6.4. Previous research has revealed that improved volume fraction predictions can be achieved through the use of a hidden layer of neurons [4]. A model search was conducted to determine the optimal number of hidden layer neurons. It was determined that optimal performance is achieved for the top system using 185 hidden layer neurons and 220 hidden layer neurons were used for the bottom system. The reconstruction results of these double-layer feed-forward neural networks are detailed in table 6.6, which includes a comparison to the results of the 8-electrode prototype.

TABLE 6.6 COMPARISON BETWEEN THE PERFORMANCE OF THE 16-ELECTRODE FREQUENCY DIVISION MULTIPLEXED IMPEDANCE TOMOGRAPHY SYSTEM AND THE 8-ELECTRODE PROTOTYPE FREQUENCY DIVISION MULTIPLEXED IMPEDANCE TOMOGRAPHY SYSTEM FOR A THREE-PHASE AIR-GRAVEL-WATER VOLUME FRACTION PREDICTION USING A DOUBLE-LAYER FEED-FORWARD NEURAL NETWORK. ALL RESULTS REPRESENT THE ABSOLUTE ERROR AND ARE ROUNDED OFF TO TWO SIGNIFICANT DIGITS.			
PERFORMANCE MEASURE	16-ELECTRODE LABORATORY-SCALE FDM IMPEDANCE TOMOGRAPHY SYSTEM		8-ELECTRODE PROTOTYPE FDM IMPEDANCE TOMOGRAPHY SYSTEM
	TOP SYSTEM	BOTTOM SYSTEM	
AIR VOLUME FRACTION ERROR (%)	1.2	1.4	7.4
GRAVEL VOLUME FRACTION ERROR (%)	1.5	1.7	4.8
WATER VOLUME FRACTION ERROR (%)	0.86	1.0	4.7
SUM VOLUME FRACTION ERROR	3.6	4.1	17

Although an improvement in the accuracy of the volume fraction predictions was achieved through the addition of a hidden layer of neurons, the results were still comparable with those of the image reconstruction.

It is evident from the results of the previous sections that a consistent difference exists between the performance of the top measurement system and the bottom measurement system. This difference is probably the result of each system having different amplification settings due to component tolerances and minor variations in the measurement electrodes. Further, it is the nature of the neural network reconstruction algorithm that this performance difference could be the result of minor variations in the training database generation for the top and bottom systems. However, the effect of these variations is not that significant since it is possible to perform a reconstruction of the top system using a neural network trained for the bottom system, and vice versa, although the reconstructed images may not be accurate. The following chapter examines the results of using neural networks trained with static data to reconstruct dynamic flow situations.

CHAPTER 7

DYNAMIC SITUATION RESULTS FOR THE DUAL-PLANE 16-ELECTRODE IMPEDANCE TOMOGRAPHY CROSS-CORRELATION SYSTEM

The theory of cross-correlation flowmeters is briefly discussed in the following section before the dynamic performance of the dual-plane 16-electrode FDM impedance tomography system is analysed.

7.1 Cross-correlation Flowmeter Fundamentals

As mentioned previously, cross-correlation flow measurement is based on obtaining the cross-correlation function of the signals obtained from two sensors spaced axially along the pipe. The time delay corresponding to the maximum value of the cross-correlation function gives an indication of the transit time between the two sensors.

The cross-correlation function of the value of $x(t)$ at time t at the upstream sensor and the value of $y(t)$ at time $t + \tau$ at the downstream sensor is calculated as follows [64]

$$R_{xy}(\tau) = \lim_{T \rightarrow \infty} \frac{1}{T} \int_0^T x(t)y(t+\tau)dt \quad (7.1)$$

The cross-spectral density function $S_{xy}(f)$ is the Fourier transform of the cross-correlation function and is calculated as follows [64]

$$S_{xy}(f) = \int_{-\infty}^{\infty} R_{xy}(\tau)e^{-j2\pi f\tau}d\tau \quad (7.2)$$

and is useful for providing a more detailed analysis of the dynamics of a flow. In particular, the cross-spectral density function enables one to determine the attenuation and delay of the different frequency components in the propagation of turbulence from the first sensor to the second [50]. Hence, the cross-spectral density function can be used to detect the existence of flow disturbances travelling at different speeds [50]. This density function is defined over all frequencies, both positive and negative. In practice, it is more convenient to work with non-negative frequencies only and so the one-sided spectral density function $G_{xy}(f)$ is defined as follows [64]

$$G_{xy}(f) = \begin{cases} 2S_{xy}(f) & \text{for } f \geq 0 \\ 0 & \text{for } f < 0 \end{cases} \quad (7.3)$$

The Fourier transform and the inverse Fourier transform provide a computationally efficient means of calculating the cross-correlation function as follows [50]: firstly, the Fourier transforms $X(k)$ and $Y(k)$ are calculated for the discrete functions $x(n)$ and $y(n)$, where $X(k) = \text{FFT}(x(n))$ and $Y(k) = \text{FFT}(y(n))$. The cross-spectral density function is then calculated as $G_{xy}(k) = X(k)Y^*(k)$ where the $*$ denotes the conjugate. Finally, the cross-correlation function is calculated using the inverse Fourier transform as $R_{xy}(n) = \text{IFFT}(G_{xy}(k))$.

Although the above technique is more computationally efficient, simplicity of solution has resulted in the cross-correlation often being calculated using the direct or point-by-point method described below. In particular, the cross-correlation results achieved in the following sections are obtained using an implementation of the point-by-point cross-correlation algorithm. Since the cross-correlation is often performed on sampled data, the sampled data form of the standard cross-correlation function is as follows [50]

$$\hat{R}_{xy}(j\Delta t) = \frac{1}{N} \sum_{n=1}^N x_n y_{n+j} \quad j = 0, 1, 2, \dots, J \quad (7.4)$$

where Δt is the sample interval, N is the number of samples and x_n is sample number n from sensor x . This equation is calculated using either point-by-point calculation or evolutionary calculation. Point-by-point cross-correlation expands the above equation into the set of individual equations shown below [50]

$$\begin{aligned} R_{xy}(0) &= x_1 y_1 + x_2 y_2 + \dots + x_N y_N \\ R_{xy}(1) &= x_1 y_2 + x_2 y_3 + \dots + x_N y_{N+1} \\ &\vdots \\ R_{xy}(J) &= x_1 y_{1+J} + x_2 y_{2+J} + \dots + x_N y_{N+J} \end{aligned} \quad (7.5)$$

Since the aim is to detect the peak of the function, it is not necessary to divide by N .

Evolutionary cross-correlation provides a means of obtaining a quick approximation of the position of the peak without having to do the complete cross-correlation calculation as is done in the point-by-point technique. The method of evolutionary calculation is described by rewriting the above equations as follows [50]

$$R_{xy}(j\Delta t) = x_1 \underbrace{\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_{1+J} \end{bmatrix}}_1 + x_2 \underbrace{\begin{bmatrix} y_2 \\ y_3 \\ \vdots \\ y_{2+J} \end{bmatrix}}_2 + \dots + x_N \begin{bmatrix} y_N \\ y_{N+1} \\ \vdots \\ y_{N+J} \end{bmatrix} \quad (7.6)$$

Clearly, if all the evolutions are included in the calculation then it is effectively the same as the point-by-point method and no speed advantages are achieved. However, a quick approximation of the cross-correlation function can be obtained by only performing the calculation for the first few evolutions. Evolution 1 and 2 are highlighted in the above equation. As more evolutions are included in the calculation, greater accuracy is achieved at the cost of increased processing time.

Since cross-correlation flowmeters are only required to measure a time delay, certain sensor properties are not critical. For example, the linearity and DC stability of the readings are unimportant [50]. In contrast, the phase delay of the transducers requires careful consideration. Specifically, the filters used in both transducers must be matched since a difference in the phase delay of the two systems would result in a constant bias to the velocity measurements. These time delay estimation errors have been investigated and it can be shown that the error in time delay measurement is a function of the difference between the two time constants T_1 and T_2 of the two planes. Assuming the two measurement planes have a first-order low-pass filter response, the error in the time delay estimation is approximately [50]

$$|\tau_a - \tau_m| \approx \frac{1}{6} (T_2 - T_1) \quad (7.7)$$

where τ_m is the measured time delay and τ_a is the actual time delay.

As mentioned previously, the bandwidth of the signals plays an important role in the cross-correlation function and specifically, in terms of the accuracy with which the peak of the cross-correlation function can be measured. The bandwidth of a cross-correlation system depends on three factors [50]:

1. the physical properties of the process itself since smaller bubbles would give rise to the generation of signals with higher frequency components as the bubbles move through the measurement space
2. the flow velocity since there are more transitions per second for a faster flow
3. the spatial filtering effect of the sensor

The spatial filtering effect often determines the maximum frequency component of the correlated signals. Further, if a digital cross-correlator is used it is important to satisfy the Nyquist sampling theorem to ensure erroneous results are not generated through aliasing [50]. It can be shown that the variance of the measured time delay τ for bandwidth-limited white noise is given by the following equation [50]

$$\text{var}(\tau) = \frac{0.038}{TB^3 \left(R_{xx}(0) / R_{nn}(0) \right)} \quad (7.8)$$

where B is the bandwidth of the system, T is the time duration over which the correlation is performed and

$R_{xx}(0) / R_{nn}(0)$ defines the mean square signal-to-noise ratio. From this it is evident that a slight decrease in the system bandwidth B will result in a sharp increase in the variance of time delay measurement thus indicating the importance of high bandwidth signals for cross-correlation [50].

Various assumptions are made when measuring the velocity field by cross-correlating tomographic information [5, 9]

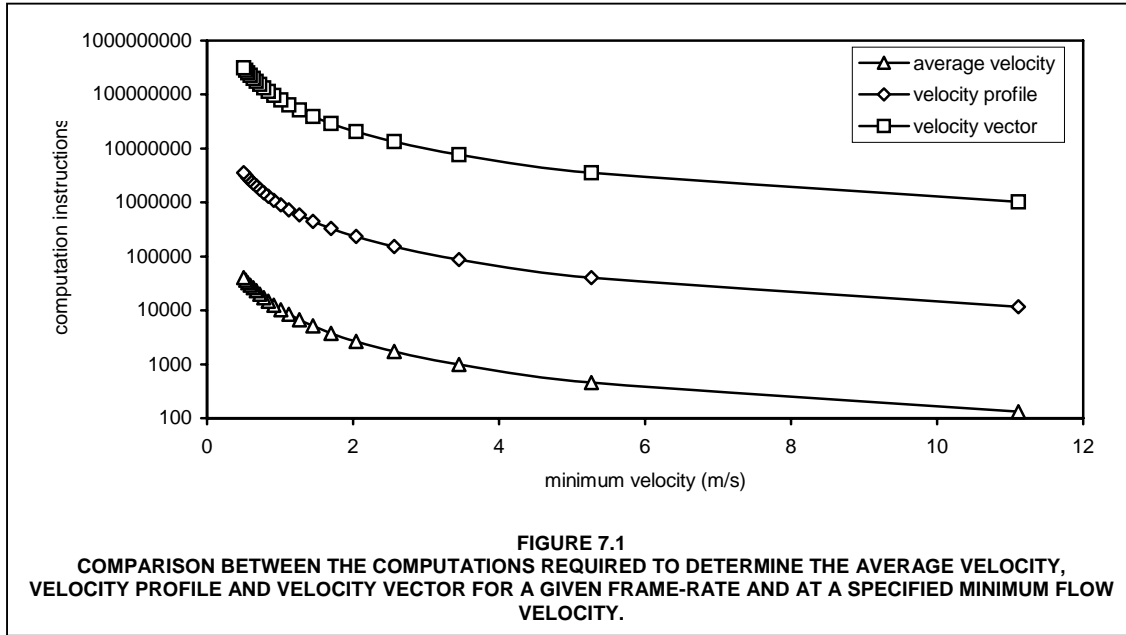
1. the size of the sensor is small relative to the separation between planes
2. there exist measurable disturbances in the flow field being investigated
3. the velocity field can be associated with the propagation velocity of these disturbances.

Clearly, an electrical impedance tomography system does not strictly satisfy assumption one. Since the electrode length is generally comparable to the plane separation, questions are raised as to how exactly the plane separation should be defined [5].

For a cross-correlation flowmeter to work correctly, the signals detected by the top and bottom systems should be a random series. These signals should represent a random modulation effect of the fluid in the sensing volume of the sensor. In a bubbly gas-liquid flow, this modulation is achieved through the random positioning of the bubbles. In contrast, a stratified or annular flow produces no such random effect, assuming that the interface is not wavy, and consequently the cross-correlation flowmeter will not work [12].

As an example of the problems associated with assumption three, consider the case of a pneumatic conveying system, where pellets of a certain material are conveyed along a pipeline using air. Here the pellets move along the pipe in a wavelike manner forming slugs. It is these slugs that are detected and correlated by the tomography system, and not the individual pellets. However, the velocity of the individual pellets cannot be directly linked to the propagation velocity of the slugs and so the cross-correlation algorithm fails to determine the true material velocity [5, 9].

In the current application, the volume fractions of the air and gravel phases are correlated to determine the average velocity of each component. This approach does not give any indication as to how the velocities of the individual components vary over the cross-section of the pipeline. However, by cross-correlating the individual pixels of two sequences of images, the individual axial velocity vectors at each pixel can be obtained, producing a velocity profile [14]. For a two-phase system, 88 correlation algorithms would be performed instead of the single correlation based on component volume fraction. Consequently, the computation required would increase by a factor of 88. In certain applications, the flow may take on spiralling forms. Consequently, every pixel in the first plane would need to be correlated with every other pixel in the second plane to give the average velocity vector of each pixel between the two planes [17]. This approach will obviously increase the computation by a further factor of 88. Figure 7.1 is a plot showing the difference in computational requirements between each of these three techniques using a point-by-point cross-correlator. The computation is defined as the number of program instructions executed. As the minimum velocity of the system decreases, so the number of frames over which the correlation is performed increases for a given frame-rate, and hence the computation increases.



One possible reduction in the computational requirements can be achieved by noting that the signals of adjacent pixels are also themselves highly correlated [59]. Consequently, it may be possible to reduce the number of pixels in the images that are correlated. Otherwise, large amounts of redundant information may be obtained.

Using the velocity profile information obtained from the cross-correlation function, it is possible to determine the mass flow rate of a particular component n within a multi-component flow as follows [50]

$$M_n = A \bar{u} W_n \rho_n \quad (7.9)$$

where A is the cross-sectional area of the pipe, \bar{u} is the mean flow velocity, W_n gives the fractional cross-sectional area of the pipeline occupied by component n and ρ_n is the density of component n . The volume fraction calculated by the neural network is effectively the same as W_n and \bar{u} is calculated by the cross-correlation of the volume fractions measured at the two planes. This equation is based on the assumptions that the flow distribution is fixed and the components in the multi-phase flow are moving at the same velocity.

For the case of components travelling at different velocities, the mass flow rate of component n is given by [50]

$$M_n = \frac{1}{T} \int_0^T \int_{x=0}^{2R} \int_{y=0}^{2R} u(x,y,t) W_n(x,y,t) dx dy dt \quad (7.10)$$

where R is the radius of the pipeline, $u(x,y,t)$ is the velocity of a particular pixel measured using the cross-correlation of the pixel values and

$$\begin{aligned} W_n(x,y,t) &= \rho_n & \text{if } \rho(x,y,t) &= \rho_n \\ W_n(x,y,t) &= 0 & \text{if } \rho(x,y,t) &\neq \rho_n \end{aligned} \quad (7.11)$$

is determined using the pipeline imaging system.

7.2 Real-time Sampling and Reconstruction Software Description

The following sections detail the design of a flow-simulation apparatus and the modifications made to the neural network reconstruction software to perform on-line real-time reconstructions.

7.2.1 Design of a flow-simulation apparatus

In order to verify the dynamic reconstruction performance of the dual-plane 16-electrode FDM impedance tomography system, it was necessary to design some apparatus that could accurately control both the speed and position of simulated masses. It was decided that stepper motors would provide this speed and position control. Figure 7.2 on page 83 is a diagram of the flow-simulation apparatus developed for this particular application.

The requirements of this flow-simulation apparatus were as follows:

- The position of the bubble must coincide with one of the standard bubble positions defined by the bubble placement system. This was necessary to ensure that optimal reconstruction results could be achieved using the neural networks trained on static data. Consequently, the bubble placement system was replicated for the dynamic simulations. It consists primarily of a positioning base, which is included as a detail in figure 7.2. The positioning base is placed in the bottom section of the laboratory-scale rig and the alignment strips in the bottom section are used to align the positioning base with the measurement sections. 20mm holes are drilled at the corners of each pixel, as can be seen in the figure detail. Before a test is conducted, the 20mm PVC pipe used to support the pulleys is inserted into one of these holes. Subsequently, the position of the bubble on the drive belt corresponds to one of the standard bubble positions used during network training.
- Ideally, the apparatus is required to simulate flow velocities up to 20m/s. Since the maximum stepper motor speed is 50rev/s, as set by the stepper driver, this places a constraint on the minimum pulley diameter. In addition, the pulley diameter is required to be a multiple of 35mm in order to ensure that the position of the bubble corresponds to a standard bubble position. Hence, a pulley diameter of 140mm was chosen, corresponding to a maximum flow velocity of 22m/s. A 3mm o-ring drive belt was used and the pulleys were machined out of polyacetal plastic. In addition, the pulley mounting blocks were machined from polyacetal plastic and were specifically designed to ensure that the spacing between the PVC pipe and the pulley resulted in the correct positioning of the bubbles.
- Two systems were designed to operate in the laboratory-scale rig at the same time. This feature is used to simulate two different bubbles moving at different velocities through the measurement sections at the same time.

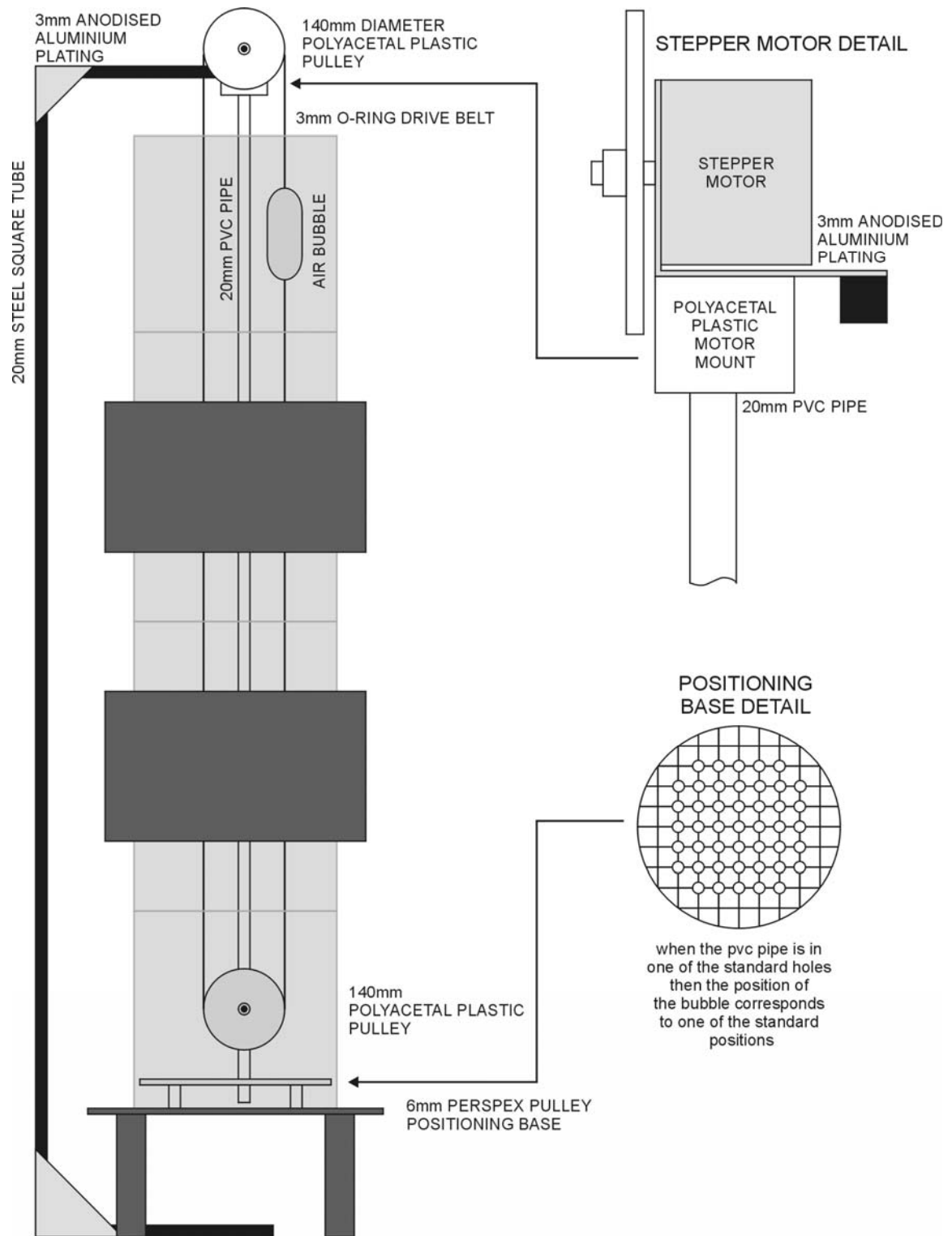


FIGURE 7.2
DIAGRAM OF THE FLOW-SIMULATION APPARATUS WITH INCLUDED DETAIL OF THE MOTOR MOUNT AND POSITIONING BASE.



FIGURE 7.3
PHOTOGRAPH OF THE FLOW-SIMULATION
APPARATUS STANDING NEXT TO THE
LABORATORY-SCALE RIG.



FIGURE 7.4
PHOTOGRAPH OF THE FLOW-SIMULATION
APPARATUS FOR THE SPECIFIC TEST OF A 0.04
VOLUME FRACTION GRAVEL AND A 0.04
VOLUME FRACTION AIR BUBBLE PASSING
THROUGH THE PIPELINE AT THE SAME TIME.

Figure 7.3 is a photograph of the flow-simulation apparatus standing next to the rig and figure 7.4 is a photograph of both flow-simulation systems before an air-gravel-water test.

The stepper motors are controlled using a stepper controller and driver system obtained from DebTech. The stepper controller panel consists of an AT6400 4-axis stepper controller, two OEM330 3A stepper drivers and one OEM530 5A stepper driver. A controller card installed in the reconstruction computer controls the AT6400 4-axis stepper controller. A Pascal device driver, provided with the controller card, was ported into Delphi to control the stepper motor for flow-simulation purposes. The complete functionality of the stepper controller was coded in a unit, which can be found in Appendix L on page 189. Specifically, this unit provides access to high-level routines that can be called without having to generate the operating system syntax used by the stepper controller.

Examples of these high-level procedures include:

- Initialise: loads a text file of initialisation data, such as the drive resolution and the stepper pulse width, and configures the AT6400 stepper controller.
- DownloadOS: the AT6400 stepper controller requires an operating system to run and this procedure downloads the operating system file to the stepper controller.
- GoDrive: initiates motion on the specified drives at the specified velocity and in the specified direction
- JogMove: jogs the specified drive in the specified direction and is used for the accurate positioning of the bubbles before a test.
- MoveFixedDist: initiates motion on the specified drives at the specified velocity and for the specified distance. This procedure is used for flow-simulation purposes where a bubble is moved over a fixed distance from the base of the rig to the top of the rig at a fixed velocity.
- StopDrive: brings all the drives to a controlled stop using the defined deceleration ramp.
- GetMotorData: accesses the fast data area of the stepper controller to determine the motor position and velocity. Since an encoder is not used, these readings are meaningless if the stepper motor fails.

Tests were conducted to determine the maximum acceleration of the stepper motors under load. Clearly, it is desirable to use the maximum possible acceleration so as to permit the simulation of higher flow velocities within the limited distance of the flow-simulation apparatus. However, due to the large diameter of the pulley, the torque of the stepper motors limited the acceleration to approximately 50rev/s². Consequently, the maximum velocity that could be simulated is 4.7m/s at the centre of the rig and the maximum average velocity between the two measurement planes is 4.2m/s. Once the bubbles are attached to the drive belt and the apparatus is inserted into the water of the laboratory-scale rig, then these maximum velocities are reduced. However, the buoyancy of the air bubble partly compensates for the friction of the water in a simulated upward flow. Specifically, the maximum velocity of the 0.01 volume fraction air bubble is 4.2m/s and the maximum velocity of the 0.04 volume fraction air bubble is 3.6m/s. In contrast, the weight of the 0.04 volume fraction gravel bubble limits the maximum upward flow velocity to 1.4m/s. In addition, the drive belt starts to slip on the pulley for the higher velocity tests. This slippage obviously limits the accuracy of a dynamic verification algorithm since the position of the bubble is not guaranteed. This is examined in greater detail in the following section.

It should be noted that the current system is unable to measure the velocity of the water component in a three-phase air-gravel-water mixture. Clearly this could be a problem for certain applications of this technology where it is important to know the individual velocities of all the components in a multi-phase mixture. However, in terms of the intended application of this system, it is only really the velocity of the gravel component that is critical. Subsequently, this limitation was largely ignored, although future research should address this problem in order to apply the technology to a wider range of applications. Because of this limitation, no attempt was made to simulate the movement of water through the measurement sections.

7.2.2 Verifying dynamically acquired data

The neural network reconstruction software includes a 'Real-time sampling' form which allows one to test the dynamic performance of the dual-plane impedance tomography system. The commented program code for this unit can be found in Appendix L on page 178. The following section will describe the functionality of this form as well as the accompanying dynamic verification form. Figure 7.5 on page 86 is a screen capture of the 'Real-time sampling' form and a full-scale version of this screen capture can be found in Appendix N on page 208.

The form is divided into two separate sections, namely the reconstruction results and the configuration section. To perform a dynamic test it is necessary to configure the system beforehand. This requires specifying the neural networks to perform the reconstructions of the top and bottom systems, the plane separation and the flowmeter factor. The flowmeter factor α relates the true mean flow velocity u_a to the velocity measured using the cross-correlation flow measurement u_c as $\alpha = \frac{u_a}{u_c}$ [50]. In certain configurations, this factor can be obtained through

calibration. In situations where calibration cannot be performed, certain analytical methods exist as a low-cost alternative to calculate the flowmeter factor [50]. Tests have shown that accurate velocity prediction results are obtained for the laboratory-scale rig with a unity flowmeter factor.

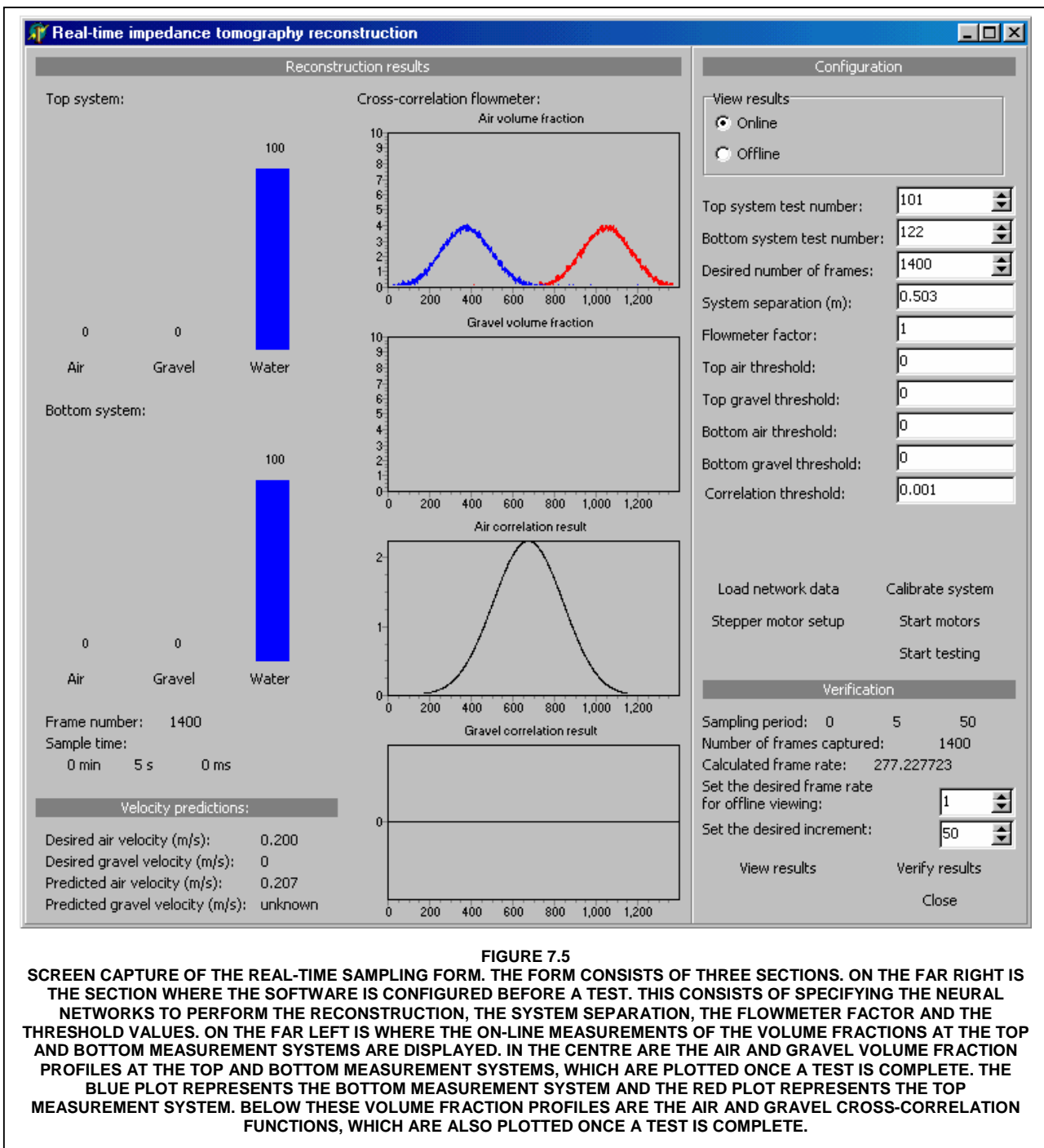


FIGURE 7.5
SCREEN CAPTURE OF THE REAL-TIME SAMPLING FORM. THE FORM CONSISTS OF THREE SECTIONS. ON THE FAR RIGHT IS THE SECTION WHERE THE SOFTWARE IS CONFIGURED BEFORE A TEST. THIS CONSISTS OF SPECIFYING THE NEURAL NETWORKS TO PERFORM THE RECONSTRUCTION, THE SYSTEM SEPARATION, THE FLOWMETER FACTOR AND THE THRESHOLD VALUES. ON THE FAR LEFT IS WHERE THE ON-LINE MEASUREMENTS OF THE VOLUME FRACTIONS AT THE TOP AND BOTTOM MEASUREMENT SYSTEMS ARE DISPLAYED. IN THE CENTRE ARE THE AIR AND GRAVEL VOLUME FRACTION PROFILES AT THE TOP AND BOTTOM MEASUREMENT SYSTEMS, WHICH ARE PLOTTED ONCE A TEST IS COMPLETE. THE BLUE PLOT REPRESENTS THE BOTTOM MEASUREMENT SYSTEM AND THE RED PLOT REPRESENTS THE TOP MEASUREMENT SYSTEM. BELOW THESE VOLUME FRACTION PROFILES ARE THE AIR AND GRAVEL CROSS-CORRELATION FUNCTIONS, WHICH ARE ALSO PLOTTED ONCE A TEST IS COMPLETE.

Cross-correlation flowmeters require a zero-flow detector because cross-correlators are incapable of measuring flow below a certain minimum value [50]. If there is no flow then the sensor signals will be predominantly electrical noise. The cross-correlator will always attempt to perform a correlation. Consequently, spurious velocity measurements will result from the correlation of this random electrical noise. Since the laboratory-scale rig does not have a zero-flow detector, it was necessary to code thresholds into the cross-correlation software. Firstly, the volume fraction thresholds are used to remove a constant output offset from the volume fraction profile, if such an offset exists. Secondly, a threshold is performed on the peak of the cross-correlation function. If the peak of the cross-correlation is not larger than the threshold then it is determined that the peak of the cross-correlation is probably the result of noise. Consequently, the phase velocity of the corresponding component is predicted as zero.

The network weights and data pre-processing are loaded for the top and bottom networks according to the specified test number when 'Load network data' is clicked. A set of calibration data is captured from the rig when 'Calibrate system' is clicked. This provides a reference point against which all future tests are compared. Specifically, these calibration voltages are subtracted from all the voltages captured during testing so that only the differences are input to the neural network reconstruction algorithm. An advantage of this approach is that it is possible to calibrate the effect of the flow-simulation apparatus in the rig. Consequently, when a test is conducted, the flow-simulation apparatus is ignored and only the bubble is detected.

Clicking on 'Stepper motor setup' displays a form that is used for the configuration of the flow-simulator apparatus. A screen capture of this form can be found in Appendix N on page 209. This form initialises the stepper controller and allows the user to specify the desired direction and velocity of the simulated bubble flow. The desired flow velocity is specified in metres per second. Using the pulley diameter and the drive resolution, this form converts the velocity and distance parameters into step units that are then used to configure the stepper controller before a test. To calculate the required pulley direction, it is necessary to indicate whether the bubble is situated on the 'inside' portion of the drive belt or the 'outside'. For example, figure 7.3 on page 84 is a photograph of the bubble on the 'outside' portion of the drive belt. In order to verify the dynamic performance of the system, all bubble simulations must start from a defined reference point. The reference starting point for an upward flow is a point marked 115mm below the bottom measurement section. This defines position zero and the positions of all other points are calculated in terms of the axial distance from this reference mark. Before a test, each bubble must be positioned so that the top of the bubble corresponds to the reference mark. The jog buttons allow the user to perform this positioning.

Returning to the 'Real-time sampling' form, the test is started by clicking 'Start testing'. This initiates the stepper motor movement and also the data capture and reconstruction for the dual-plane impedance tomography system. The on-line reconstruction results for the top and bottom system are displayed in the left-hand portion of the reconstruction results section. Once the test is complete, the volume fraction profiles for the air and gravel phases are plotted and the cross-correlation performed. In these charts, the red series is the volume fraction profile of the top system while the blue series is the volume fraction profile of the bottom system. As can be seen in figure 7.5 on page 86, the air bubble was first detected at the bottom system before being detected at the top system, as would be expected for an upward flow. The following is a brief discussion as to how the software performs an on-line reconstruction at a rate of 280frames/s.

As mentioned previously, the data is captured in sets of 50 frames and is controlled by the EDR device driver as a background process. This creates an interval during which the reconstructions for the previous set of 50 frames can be performed and their results displayed. To ensure a balance is maintained between the two processes, a 'busysampling' and 'busyreconstruction' flag are used. These flags are defined as follows: 'busyreconstruction' is true while the program is performing the reconstructions for the previous set of 50 frames and 'busysampling' is true while the next set of 50 frames is being captured. It is important to note that although the data is captured in sets of 50 frames, a constant sampling rate is still maintained. This is an important requirement if the system is to be used for on-line control purposes. Specifically, the time taken to perform the reconstructions for the previous set of 50 frames is typically shorter than the time taken to capture the next set of 50 frames. Subsequently, the sampling process can start the capture of the next set of 50 frames as soon as it finishes the capture of the current set of 50 frames, since it does not have to wait for the reconstruction process to complete. In this way, both

tomography systems are sampled at a constant frame-rate. This will be examined in greater detail in section 7.4.3 on page 106. Overall, the operation of the on-line real-time reconstruction software can be explained using the following pseudo-code:

```
Start the capture of the first set of 50 frames
Set busysampling to true
WAIT until sampling completes
Start the capture of the next set of 50 frames
Set busysampling to true
LOOP   WHILE busysampling is true DO
        Start reconstructions for previous set of 50 frames
        Set busyreconstruction to true
    IF busysampling is false but busyreconstruction still true THEN
        Complete reconstructions for previous set of 50 frames
    ELSE IF busysampling is false and busyreconstruction is false THEN
        Start the capture of the next set of 50 frames
        Set busysampling to true
    ELSE IF busysampling still true THEN
        WAIT until the sampling completes
        Start the capture of the next set of 50 frames
        Set busysampling to true
REPEAT LOOP while there are still more frames to be captured
```

The use of these flags ensures that a consistent timing relationship is maintained between the sampling process and the reconstruction process.

For both image reconstruction and volume fraction prediction neural networks, the correlation is performed on the volume fraction profiles of the air and gravel phases at the top and bottom systems. A standard point-by-point cross-correlation is performed. For the cross-correlation calculation, it is necessary to include values of one of the signals beyond the measurement period T . The standard solution to this problem is to extend the appropriate data record for a further period of T seconds where $x(t) = 0$ for $t > T$ [64]. The peak of each of the cross-correlations is calculated and this corresponds to the number of frames delay between the top and bottom systems. This is then converted into a transit time using the estimated frame-rate. In particular, a system timer keeps track of the total time taken to perform the specified number of frame captures and reconstructions and hence is able to estimate the on-line frame-rate. Using these transit times and the specified plane separation, the software proceeds to calculate the velocities of the air and gravel phases. A comparison is also made to the desired phase velocities as specified when the stepper controller was configured.

Since the positions of the bubbles are exactly controlled by the stepper controller, it is possible to perform a comparison between the measured dynamic response of the dual-plane impedance tomography system and the desired response. Clicking on 'Verify results' displays a form that is used to perform this dynamic comparison and a screen capture of this form is included in figure 7.6 on page 89. A full-scale version of this screen capture is included in Appendix N on page 210. The plots of the different volume fraction profiles of the air and gravel phases for both the top and bottom system are displayed.

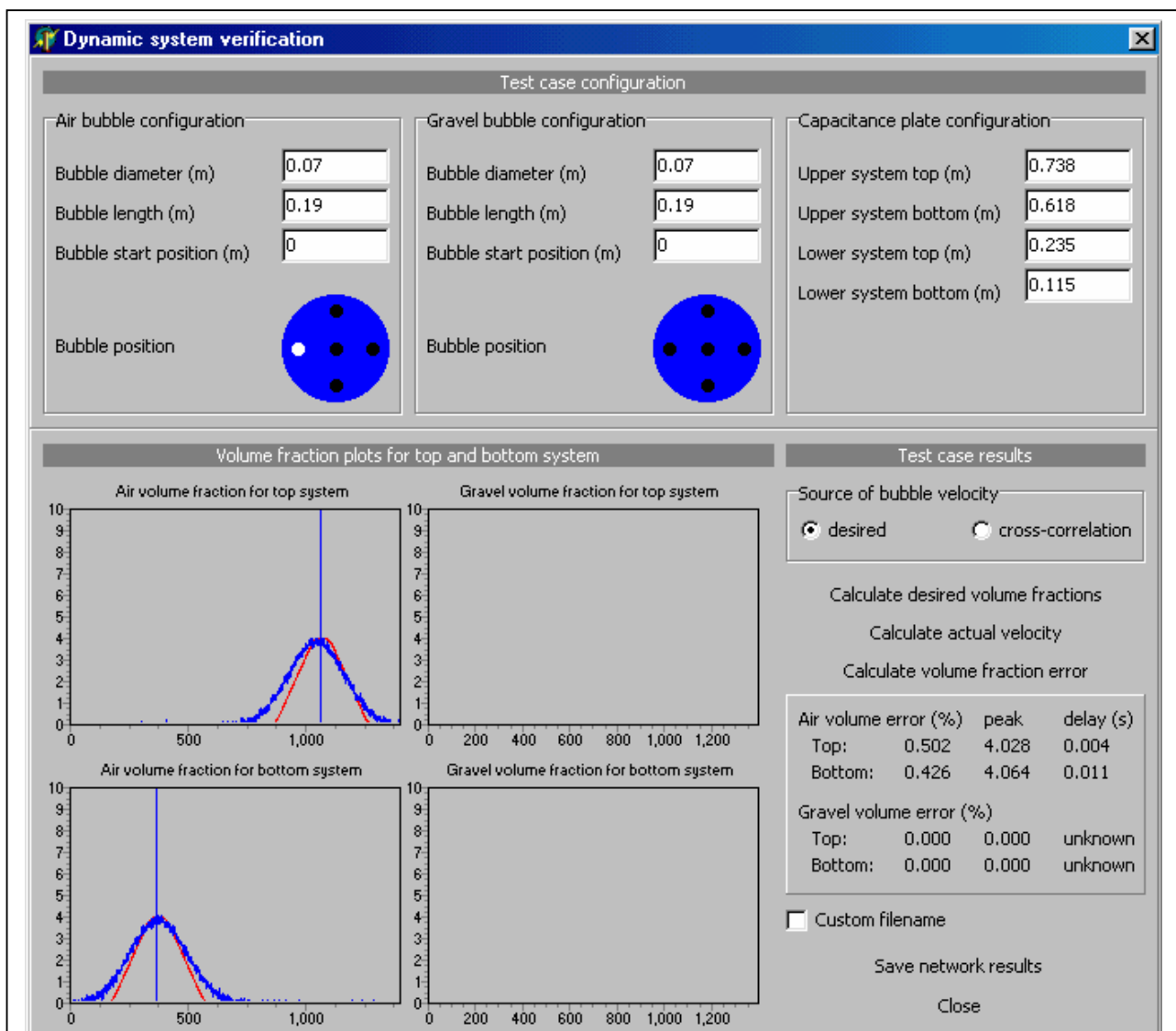


FIGURE 7.6
SCREEN CAPTURE OF THE DYNAMIC VERIFICATION FORM. THE VOLUME FRACTION PROFILES OF THE AIR AND GRAVEL PHASES ARE PLOTTED AT THE TOP AND BOTTOM SYSTEM IN BLUE. USING THE SPECIFIED BUBBLE DIMENSIONS, THE DESIRED VOLUME FRACTION PROFILES OF THE AIR AND GRAVEL PHASES AT THE TOP AND BOTTOM SYSTEMS CAN BE CALCULATED AND PLOTTED IN RED. THE ABOVE SCREEN CAPTURE IS OF A 0.04 VOLUME FRACTION AIR BUBBLE SITUATED NEAR THE LEFT-HAND EDGE OF THE RIG. THE DYNAMIC VOLUME FRACTION ERROR IS CALCULATED AS THE MEAN ABSOLUTE ERROR BETWEEN THE DESIRED AND MEASURED VOLUME FRACTION PROFILES.

To calculate the desired volume fraction profiles, the dimensions of the bubbles used in the test must be specified, as well as the positions of the top and bottom measurement electrodes relative to the reference marker discussed earlier. Using this information, the software is able to calculate the desired volume fractions as follows: since the frame-rate is known, the sample time corresponding to a particular frame number can be calculated. Then, since the acceleration and velocity of the bubble is known, the position of the bubble can be calculated for the corresponding frames. Since the position and size of the bubble is known, and the position of the measurement electrodes has been specified, the desired volume fraction for a particular frame can be calculated. In particular, the volume fractions are calculated as the fraction of the bubble within the cylindrical volume of the measurement electrodes. These desired volume fraction profiles are plotted in red and the measured volume fraction profiles in blue. The volume fraction error for the dynamic performance of the system can now be calculated.

The dynamic volume fraction error is the mean absolute error between the desired volume fraction profile and the measured volume fraction profile. Since the desired volume fraction is zero for a large proportion of the time, the mean absolute error is only calculated over the region where the bubble is within the measurement section so as to get a more representative result. An additional complication arose as a result of the slippage between the stepper motor pulley and the drive belt. Consequently, the position of the desired volume fraction profile did not coincide with the measured volume fraction profile as can be seen in figure 7.6 on page 89. To quantify the amount of slippage between the drive belt and pulley, a 0.04 volume fraction air bubble was positioned at the top of the rig and a mark was made on the belt and the pulley at the same point. The flow-simulation apparatus was then instructed to move the bubble down for one metre and then raise the bubble by one metre so that it would return to its original position. Observation of the marks on the belt and pulley after the movement showed that the belt had slipped by, on average, 150mm with respect to the pulley. It was also noted that the pulley had returned to its original position proving that the slippage was not the result of stepper motor slip. Unfortunately, the slippage between the pulley and drive belt meant that no valid information could be obtained regarding the time delay dynamic response of the dual-plane impedance tomography system. A volume fraction error could still be calculated provided software compensation corrected for the slippage.

To achieve a more representative dynamic volume fraction error, the desired volume fraction profile was shifted so as to coincide with the measured volume fraction profile to allow for drive belt slippage. The amount of shift was calculated as the number of frames between the peak of the desired volume fraction profile and the peak of the measured volume fraction profile.

7.2.3 Cross-plane interference problem

Cross-plane interference, if not accurately modelled, will result in a general loss of accuracy and can severely affect the reconstruction process [49]. Further, cross-talk between sensors can distort the cross-correlation function making accurate determination of the peak more difficult [57]. The standard technique used to prevent cross-plane interference is the method of interleaving. This ensures that only one sensing array is activated at any time, thus effectively eliminating cross-talk. For the adjacent measurement protocol in resistance tomography, the operation of the interleaving data acquisition technique is as follows [43]

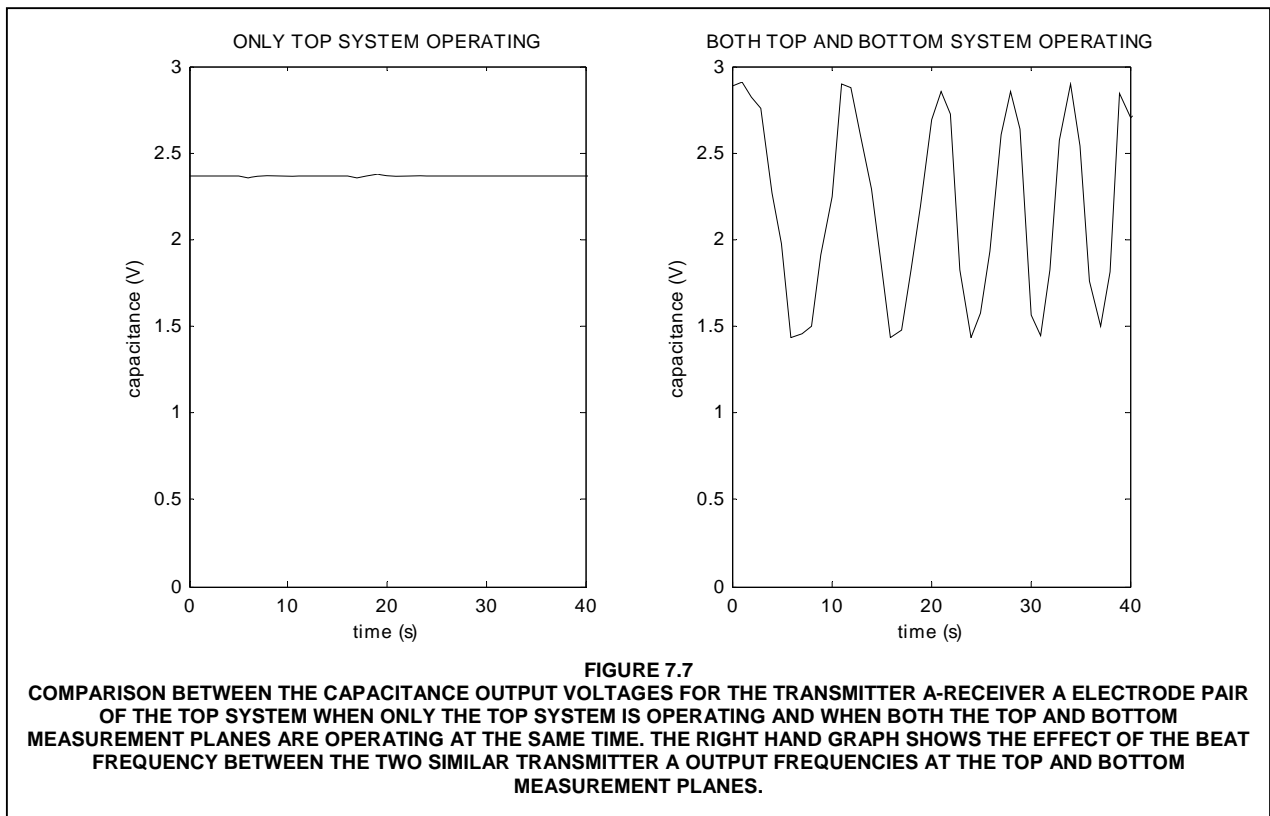
1. current is injected between electrode 1 and 2 for system 1 and the potential differences are measured on the remaining electrodes of system 1
2. current is injected between electrode 1 and 2 for system 2 and the potential differences are measured on the remaining electrodes of system 2
3. current is injected between electrode 2 and 3 for system 1 and the potential differences are measured on the remaining electrodes of system 1

and so on. By using the interleaving data collection protocol, a smaller separation between the electrode planes can be achieved [10]. As mentioned previously, this will improve the performance of the cross-correlator, especially for flows that are highly skewed [10]. The disadvantage of this technique is that the data acquisition rate is effectively half [12].

Various factors determine the selection of the spacing between the two systems. If the systems are close then the cross-correlation has a well-defined and large correlation peak enabling accurate measurement of the transit delay. In addition, the cross covariance between the two planes will be higher since the properties of the turbulence would have remained fairly consistent from the upstream to the downstream sensor [50]. However, in systems where the sensors are not point electrodes, such as capacitance tomography systems, it is difficult to define what the sensor

separation should be since the length of the electrodes are comparable to the system separation. Consequently, a large separation is required in these situations to reduce this uncertainty. Between these two extremes an optimal value for system separation exists and has been shown to be between three to four pipe diameters [50]. For the current application, it was deemed impractical to separate the two planes by three to four pipe diameters; the maximum possible separation was 1.4 pipe diameters. Therefore alternate techniques were tested in an attempt to reduce the cross-plane interference.

Tests revealed that the top and bottom measurement systems interfere when both are operating simultaneously. The effect of this interference can be explained as follows: both planes are using the same set of eight transmitter frequencies. However, each plane has its own frequency generator board. Consequently, the frequencies generated are not exactly the same. As an example, receiver A of the top system receives a frequency component from transmitter A of the top system at frequency f , and a component from transmitter A of the bottom system at frequency $f+x$, where x is typically small. Subsequently, the difference frequency x is produced at the output of the multiplication stage of the synchronous detector and passes through the low-pass filters to appear on the output as a substantial ripple component. Figure 7.7 compares the capacitance voltage output for this transmitter A-receiver A electrode pair for the top system when only the top system is operating and when both the top and bottom systems are operating at the same time.



A grounded band between the two planes of measurement electrodes has been used to reduce cross-plane interference in capacitance tomography systems [14]. The same has also been used in resistance tomography systems. A galvanised steel disk was inserted between two rubber insertion gaskets at the flange joining the top and bottom measurement planes. The disk protruded slightly into the pipeline and hence made electrical contact with the water. It was grounded and the interference analysis test was repeated. By introducing a grounded band between the two measurement planes, the ripple on the output voltages was substantially reduced. To reduce the interference even further, the top portion of the laboratory-scale rig was inserted between the top and bottom measurement sections as a spacer to increase the plane separation to 2.4 pipe diameters. In addition, a grounded disk was inserted at the flange joining the top measurement section to the spacer and at the flange joining the spacer to the bottom measurement section. The effect of these modifications was to reduce the interference even further. Table 7.1 compares the average output voltage ripple of both the capacitance and conductance readings for the different configurations tested.

TABLE 7.1
COMPARISON BETWEEN THE AVERAGE PEAK-TO-PEAK OUTPUT RIPPLE OF THE CAPACITANCE AND CONDUCTANCE READINGS FOR THE DIFFERENT CONFIGURATIONS TESTED. INCLUDED IS THE OUTPUT RIPPLE MEASURED WHEN SYNCHRONISED TRANSMITTERS ARE USED.

	ORIGINAL CONFIGURATION WITH ONLY ONE PLANE OPERATING	ORIGINAL CONFIGURATION WITH BOTH PLANES OPERATING	GROUNDING DISK BETWEEN PLANES	TWO GROUNDED DISKS AND SPACER BETWEEN PLANES	SYNCHRONISED TRANSMITTERS
CAPACITANCE	21mV pk-to-pk	1.809V pk-to-pk	354mV pk-to-pk	58mV pk-to-pk	31mV pk-to-pk
CONDUCTANCE	14mV pk-to-pk	1.248V pk-to-pk	250mV pk-to-pk	39mV pk-to-pk	28mV pk-to-pk

Although these different pipeline configurations reduced the interference between the top and bottom measurement planes considerably, the magnitude of the output peak-to-peak ripple was still twice as large as when only one measurement plane was operating, as is evident in table 7.1. However, further analysis revealed a simple solution to the problem of interference in a dual-plane FDM impedance tomography system. Essentially, the ripple on the output voltages is a result of the small differences between the operating frequencies of the two planes.

Consequently, if both systems were to use a common frequency generator board, then this output ripple would not be produced. This is the concept behind synchronised transmitters, where one frequency generator board is used for both systems. Using this technique, it was possible to remove the spacers and grounded disks and still achieve an output ripple comparable to that when only a single plane was operating, as is evident in table 7.1.

Subsequently, all further dynamic dual-plane tests were performed using synchronised transmitters.

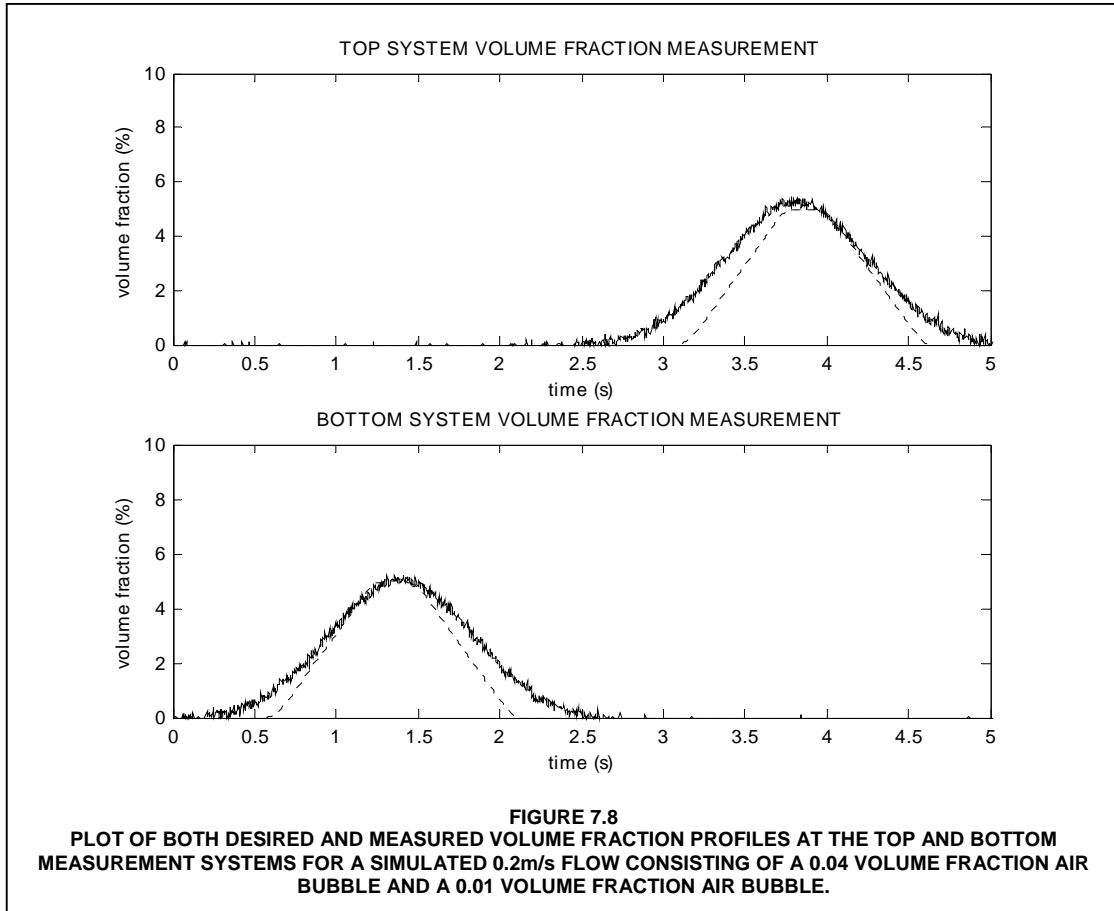
One drawback of this technique is that the receivers for a particular measurement plane are still receiving signals from the transmitters of the other measurement plane. Hence, if a large air bubble were to move through the bottom measurement plane, a change would also be detected at the top measurement plane because of the change to the interference component from the bottom measurement plane. Ideally, an FDM impedance tomography system would use a different set of frequencies for the two measurement sections ensuring that even if signals from the second plane are received at the first plane, the synchronous detectors of the first plane will reject them. However, because of the nature of square waves, it was only possible to find a single set of eight transmitter frequencies that gave adequate performance, as discussed earlier. Consequently, this cross-plane interference technique could not be employed but it should certainly be considered for the dual-plane sine wave excitation system currently being developed [62].

7.3 Two-phase Air-Water Reconstruction Results

To get a better assessment of the performance of the dual-plane 16-electrode FDM impedance tomography system, initial tests were conducted for the simpler configuration of a two-phase air-water reconstruction. Although the emphasis of the research is the development of a mass flow meter, which requires accurate volume fraction measurements, image reconstructions were also performed at various stages during the testing as a means of verifying the correct operation of the tomography systems. Where appropriate, sequences of screen captures for specific tests will be included in the following sections.

7.3.1 Real-time volume fraction prediction and image reconstruction results

The real-time dynamic performance of the dual-plane 16-electrode FDM impedance tomography system was tested using different air bubble configurations. For all tests conducted, the measured volume fraction profile was compared to the desired volume fraction profile and the dynamic volume fraction error was calculated. Figure 7.8 is a plot of the results for such a test. Specifically, this test was for a 0.04 volume fraction air bubble positioned near the left-hand edge of the rig and a 0.01 volume fraction air bubble positioned near the right-hand edge of the rig. Both bubbles were set to move upward at 0.2m/s and both bubbles were set to start moving at the same time. Hence, the volume fraction of the air phase should peak at 0.05 when the bubbles were situated directly within the measurement volume. In figure 7.8, the solid line is the volume fraction profile measured by the neural network reconstruction algorithm and the dashed line is the desired volume fraction profile calculated according to the bubble sizes and flow-simulation apparatus configuration.



As expected, the bubbles are detected at the bottom measurement plane before they are detected at the top measurement plane. In addition, it is evident that when the bubbles pass through the bottom measurement plane they are not detected at the top measurement plane thus proving that minimal interference exists between the two measurement planes. However, the decay of the measured volume fraction at the bottom plane and the increase of the measured volume fraction at the top plane are slower than the increase of the measured volume fraction at the bottom plane or the decay of the measured volume fraction at the top plane. This represents the region where the bubbles are situated between the two measurement planes. This change is probably a consequence of signals from the top plane being detected at the bottom plane and vice versa. Further, it is observed that a good correlation exists between the measured volume fraction profile and the desired volume fraction profile. As is evident in figure 7.8 on page 93, a slight offset exists between the measured and desired volume fraction profiles at the top measurement plane. As explained earlier, this is a result of the slippage between the drive belt and the pulley of the flow-simulation apparatus. Since the effect is cumulative as the bubbles move further up the pipeline, so the difference in position at the top measurement plane is more noticeable than at the bottom measurement plane. The average dynamic volume fraction error was calculated at 0.68% for the top measurement plane and 0.58% for the bottom measurement plane using a single-layer feed-forward volume fraction predictor neural network. These results represent the absolute error and are rounded off to two significant digits.

Figure 7.9 on page 95 is a sequence of screen captures of the image reconstruction results for this test. It is important to note that these reconstructions were performed in real-time. This is in contrast to standard TDM impedance tomography systems where the required number of frames are captured first and the reconstruction and analysis is then performed off-line afterwards. In each screen capture, the top frame represents the top measurement plane and the bottom frame represents the bottom measurement plane. The order of the frames is from left to right and then top to bottom. Due to the large number of frames captured during testing, only every 50th frame is included in figure 7.9. As is evident in figure 7.9, the 0.04 volume fraction air bubble is generally detected first and last as the pair of bubbles move through a measurement section. In addition, when the 0.04 volume fraction air bubble is not completely within the measurement volume of the system, it is detected as a smaller bubble. As the bubble enters the measurement volume, so the size of the bubble increases, until the desired 0.04 volume fraction air bubble is predicted. This raises questions regarding what effect the length of the bubble has on the volume fraction measurement accuracy.

Previously, all training and test data considered only bubbles that extended beyond the axial length of the measurement electrodes. For the laboratory-scale rig, the measurement plates are 120mm long. Consequently, all bubbles simulated were 190mm long. Tests were conducted to determine what effect shorter bubbles would have on the accuracy of the volume fraction measurement. Specifically, the dynamic performance of the top measurement plane was tested with air bubbles 60mm and 90mm long. Tests were also conducted for the standard air bubble length of 190mm. For each test, the peak of the measured volume fraction profile was determined and the results averaged over a number of tests.

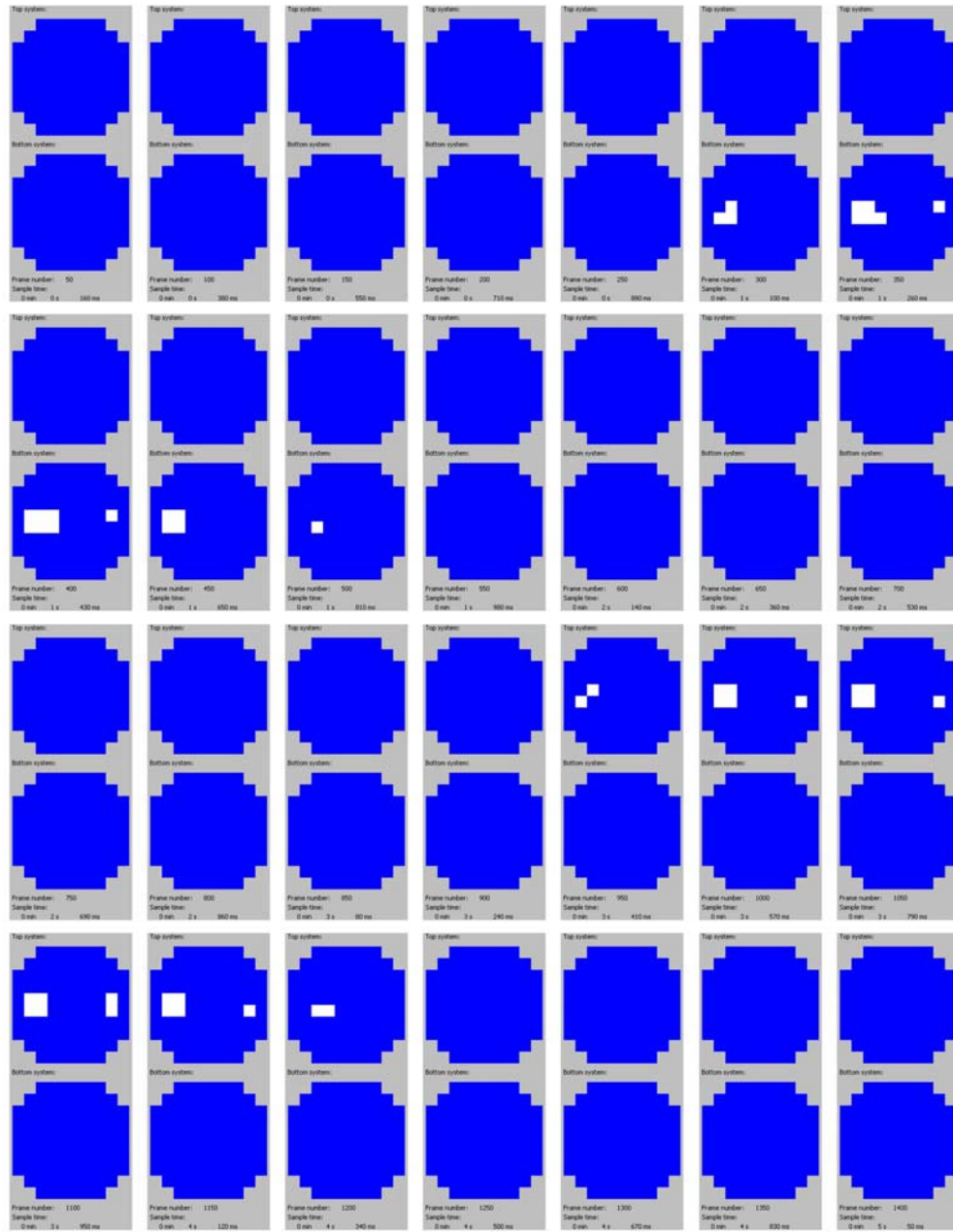


FIGURE 7.9

SCREEN CAPTURES OF A 0.04 VOLUME FRACTION AIR BUBBLE NEAR THE LEFT-HAND EDGE OF THE RIG AND A 0.01 VOLUME FRACTION AIR BUBBLE NEAR THE RIGHT-HAND EDGE OF THE RIG MOVING THROUGH THE MEASUREMENT PLANES AT THE SAME TIME. THESE IMAGE RECONSTRUCTIONS WERE PERFORMED ON-LINE AND THEIR RESULTS DISPLAYED IN REAL-TIME. IN EACH SCREEN CAPTURE, THE TOP FRAME REPRESENTS THE TOP MEASUREMENT PLANE AND THE BOTTOM FRAME REPRESENTS THE BOTTOM MEASUREMENT PLANE. THE ORDER OF THE FRAMES IS FROM LEFT TO RIGHT AND THEN TOP TO BOTTOM. DUE TO THE LARGE NUMBER OF FRAMES CAPTURED DURING TESTING, ONLY EVERY 50TH FRAME IS INCLUDED. AS BEFORE, THE BLACK IS THE WATER PHASE, THE WHITE IS THE AIR PHASE AND THE LIGHT GREY REPRESENTS THE REGION OUTSIDE THE PIPELINE.

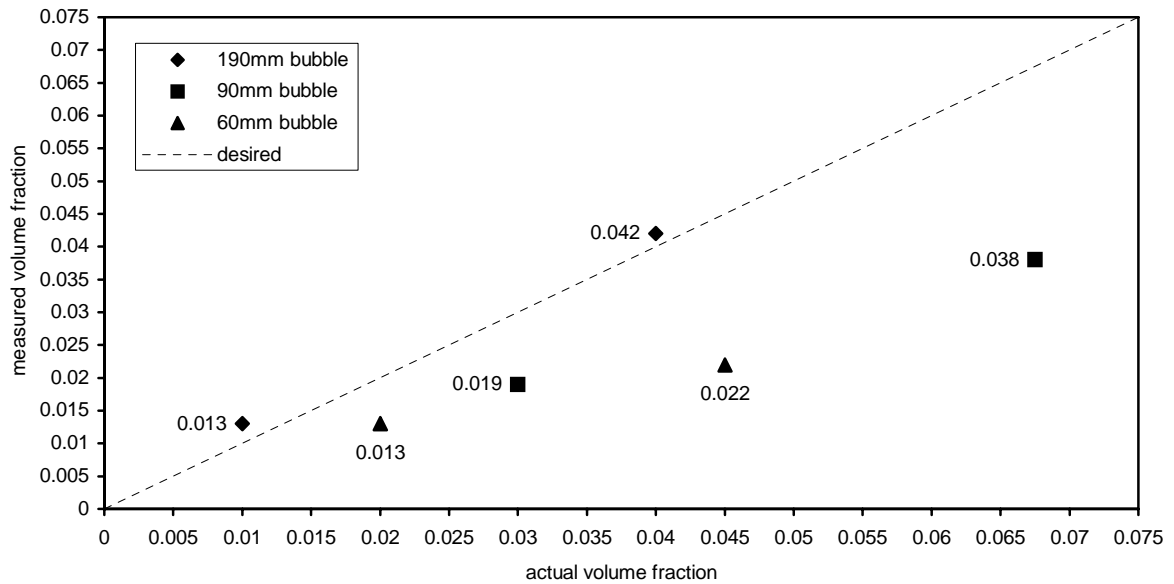


FIGURE 7.10
EFFECT OF THE BUBBLE LENGTH ON THE VOLUME FRACTION MEASUREMENT ACCURACY.

Figure 7.10 compares the actual volume fractions to the measured volume fractions for each of the three bubble sizes. As can be seen, the system accurately measures the volume fractions of the 190mm bubbles. However, as the length of the bubble decreases, so the accuracy of the corresponding volume fraction measurement decreases and the readings deviate further from the desired plot. The accuracy of these results could possibly be improved through the inclusion of shorter bubbles in the training database, although this assumption has not been verified. Table 7.2 compares the dynamic volume fraction error for each of the three bubble sizes at the top and bottom measurement planes.

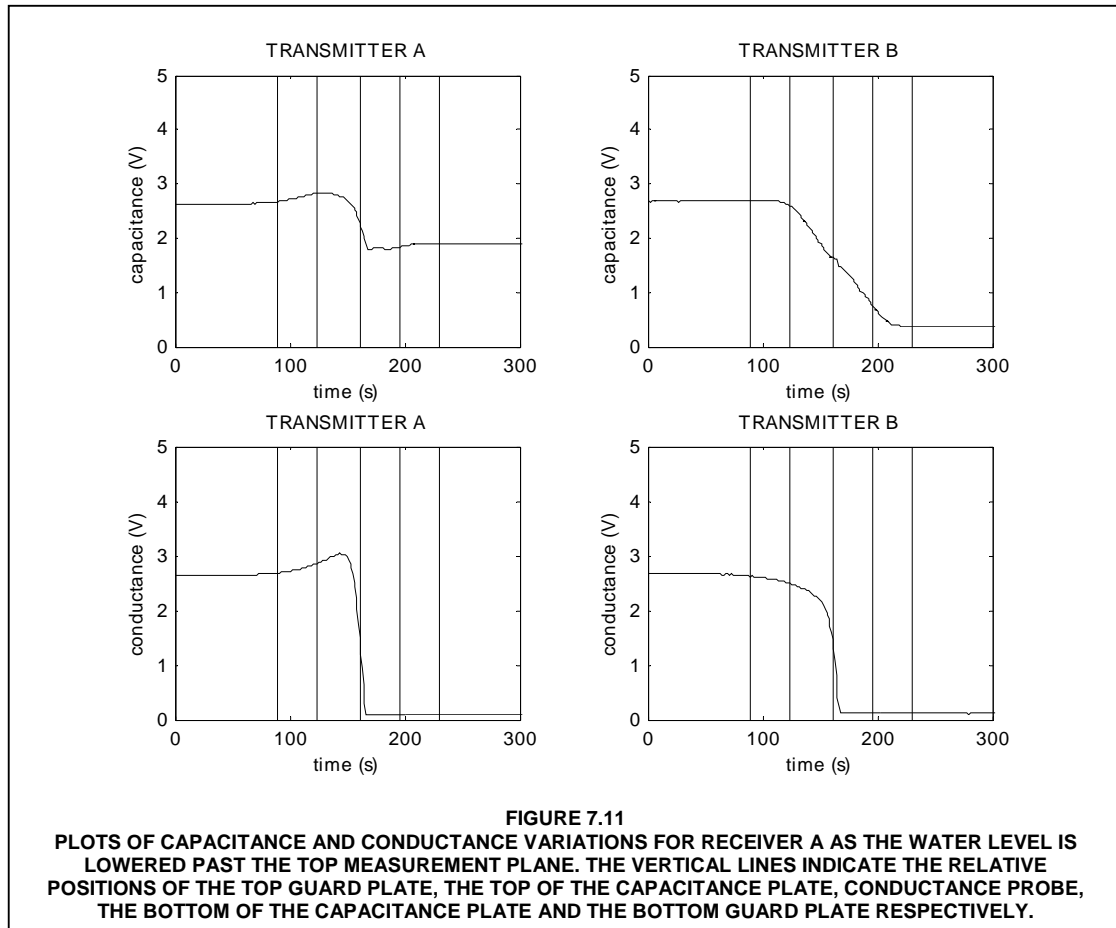
TABLE 7.2
COMPARISON BETWEEN THE AVERAGE DYNAMIC VOLUME FRACTION ERRORS FOR THE 60mm, 90mm AND 190mm BUBBLE RESPECTIVELY. ALL RESULTS REPRESENT THE ABSOLUTE ERROR AND ARE ROUNDED OFF TO TWO SIGNIFICANT DIGITS.

	DYNAMIC VOLUME FRACTION ERROR FOR SPECIFIC BUBBLE LENGTH (%)		
	60mm	90mm	190mm
TOP MEASUREMENT PLANE	1.1	1.1	0.36
BOTTOM MEASUREMENT PLANE	1.1	1.2	0.31

7.3.2 Reconstruction results for an air plug passing through the measurement section

As discussed previously, the bubbles used in training database generation were not allowed to come into contact with the conductance probes. Although this has increased the resolution at the centre of the pipeline, it was not known how the system would react to having one or all 16 conductance probes temporarily immersed in air. In order to simulate the presence of this air plug, it was decided that the water level in the rig should be lowered past the level of the conductance probes while performing an on-line real-time reconstruction. Although these tests were only conducted for the top measurement plane, it is believed that similar results would be achieved if the test were repeated for the bottom measurement plane.

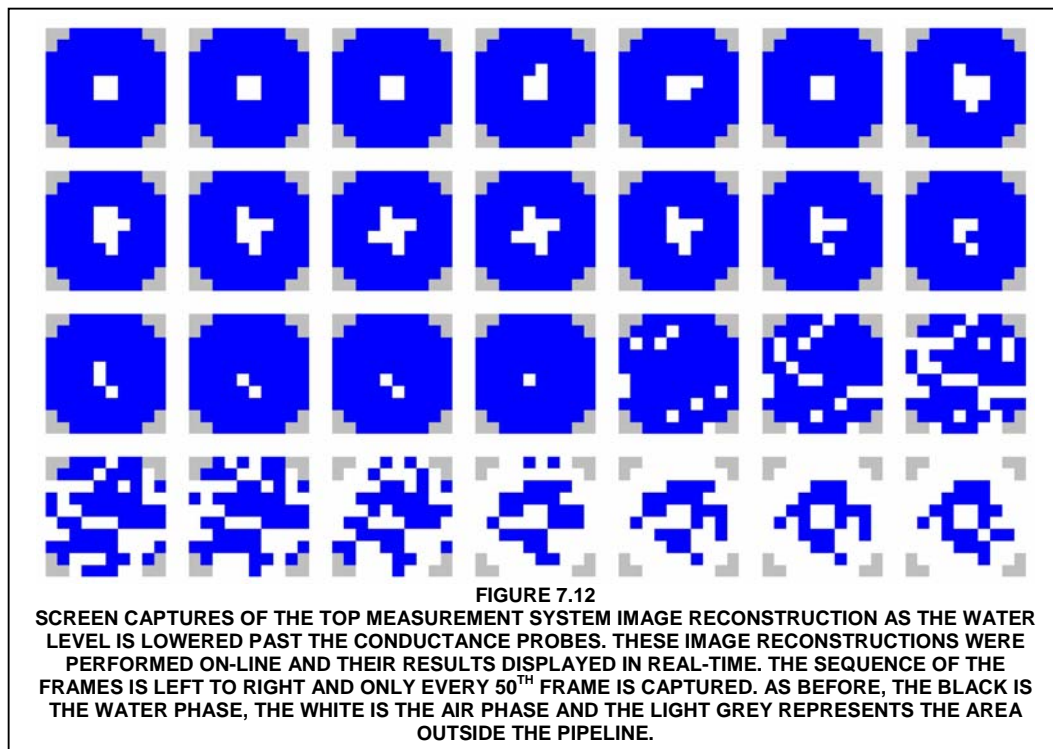
This test should also provide a useful assessment of the axial guard electrodes situated above and below the central plane of measurement electrodes. In theory, the capacitance measurements should only be dependent on the material within the volume enclosed by the measurement electrodes. Subsequently, there should be no change in the capacitance readings as the water level in the pipeline drops, until it reaches the top of the measurement electrodes. If the capacitance readings change while the water level is within the region of the guard electrodes then the guard electrodes are not operating correctly. In addition, there should be no sudden changes resulting from the water level dropping below the ring of conductance probes. In contrast, the conductance readings should decrease rapidly to zero when the water level passes the ring of conductance probes. Plots of the capacitance and conductance variations between transmitter A and receiver A and between transmitter B and receiver A are included in figure 7.11. Superimposed on these plots are the relative positions of the guard electrodes, measurement electrode and conductance probe.



It is interesting to note that the response of the adjacent transmitter-receiver electrode pairs are different to those of the electrode pairs that are spaced further apart. This phenomenon is observed for both the capacitance and conductance readings. With adjacent transmitter-receiver electrode pairs, both the capacitance and conductance readings increase before decreasing, as is evident in figure 7.11. This may be related to the phenomenon discussed earlier where the introduction of an air bubble actually results in an increase in certain capacitance and conductance readings, even though the readings are expected to decrease. In contrast, the transmitter-receiver electrode pairs that are spaced further apart behave more as expected. For example, the capacitance reading of the transmitter B-receiver A electrode pair only starts to decrease when the water level drops below the top of the measurement electrode. In addition, there is no noticeable change to the capacitance reading as the water level

drops below the conductance probes. In contrast, the conductance reading decreases almost immediately to zero as the water level drops below the level of the conductance probes for the transmitter B-receiver A electrode pair.

A sequence of screen captures were taken of the top measurement plane as the water level was lowered and these screen captures are displayed in figure 7.12. The sequence of screen captures is from left to right and only every 50th frame is captured. As can be seen, the drop in the water level is initially detected as an air bubble at the centre of the pipeline. As the water level drops, so the neural network image reconstruction results become more distorted until the water level has dropped below the conductance probes and the reconstructions stabilise. Further, it is observed that when the conductance probes are completely immersed in air, the reconstructed images do not reflect this fact and instead retain a central disk of water. This distortion in the reconstruction is probably the result of the training database not including an all-air data point, although this assumption has not been verified. Further, it is not known whether the inclusion of such a point would have a detrimental effect on the reconstruction resolution.



7.3.3 Reconstruction results for a homogenous bubble flow

The dual-plane 16-electrode FDM impedance tomography system has only been trained and tested with contiguous polystyrene foam or gravel masses that are larger than the resolution of the system. However, the flow patterns in the intended application are likely to be distributed three-phase mixtures and it is not known how a system trained on contiguous phases would respond to such a multi-phase flow mixture. To obtain an initial answer to this question, it was decided that the system should be tested on a homogenous bubble flow, since the bubbles in such a flow are smaller than the system resolution. It is important to remember that although the tomography system may not be able to resolve the individual air bubbles within a particular area, it should provide a ratio of the phases within that area [65].

The specific flow pattern achieved in a bubble column depends on the superficial velocity of the gas, properties of the liquid phase, sparger design, column diameter and height of dispersion [66]. Simply by increasing the

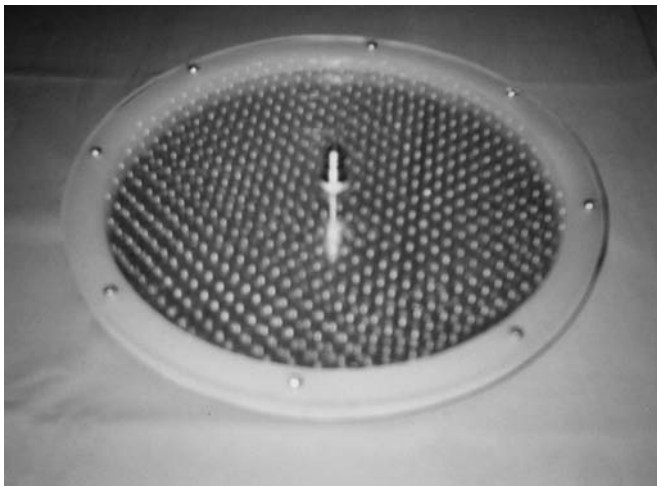


FIGURE 7.13
PHOTOGRAPH OF THE SPARGER USED TO GENERATE A
HOMOGENOUS BUBBLE FLOW IN THE LABORATORY-SCALE RIG.

superficial velocity of the gas phase, the observed pattern changes from a homogenous bubble flow to a transition regime and then a turbulent churning flow [66, 67]. A sparger was designed for the simulation of a homogenous bubble flow. It consists of two disks separated by a ring spacer and the top disk is drilled with a sequence of 1mm holes. A quick-coupler is used to connect the sparger to an air hose. This air hose connects to the regulator of an air compressor. Simply by varying the regulator pressure, different flow patterns can be generated. Figure 7.13 is a photograph of the sparger designed for the generation of a homogenous bubble flow.

The real-time volume fraction measurement was started at the same time as when the valve on the regulator was opened. Initially, no bubbles are generated. However, once the sparger has filled with air, a large surge of air bubbles is generated in the rig. This initial surge can be seen in figure 7.14, which shows the measured air volume fraction profiles during the test. As expected, the surge is detected at the bottom measurement plane before being detected at the top measurement plane.

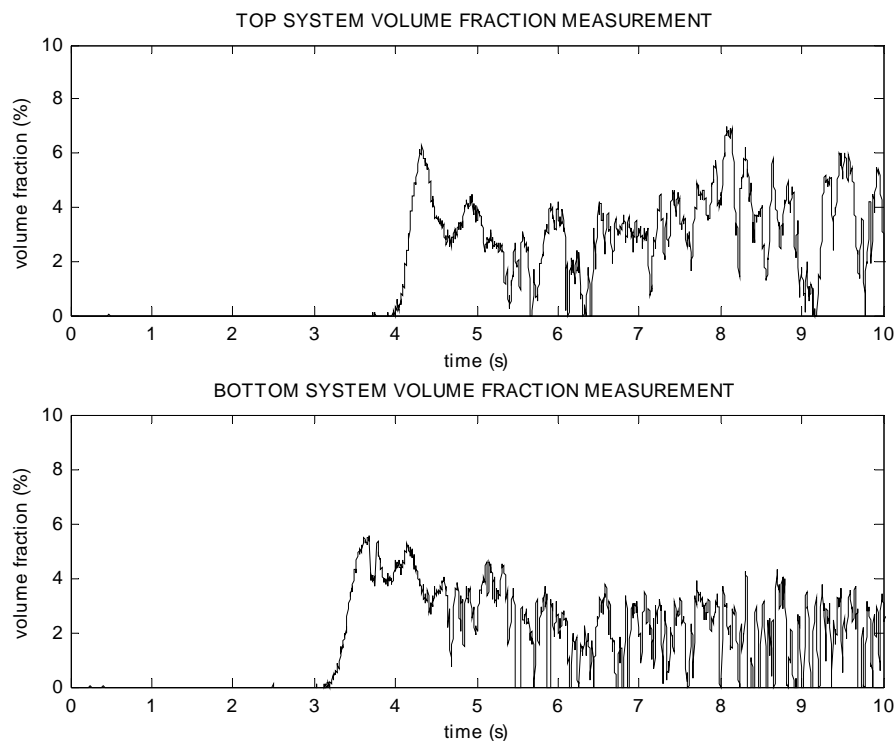


FIGURE 7.14
PLOT OF THE AIR VOLUME FRACTION MEASUREMENTS AT THE TOP AND BOTTOM MEASUREMENT
PLANES FOR A HOMOGENOUS BUBBLE FLOW.

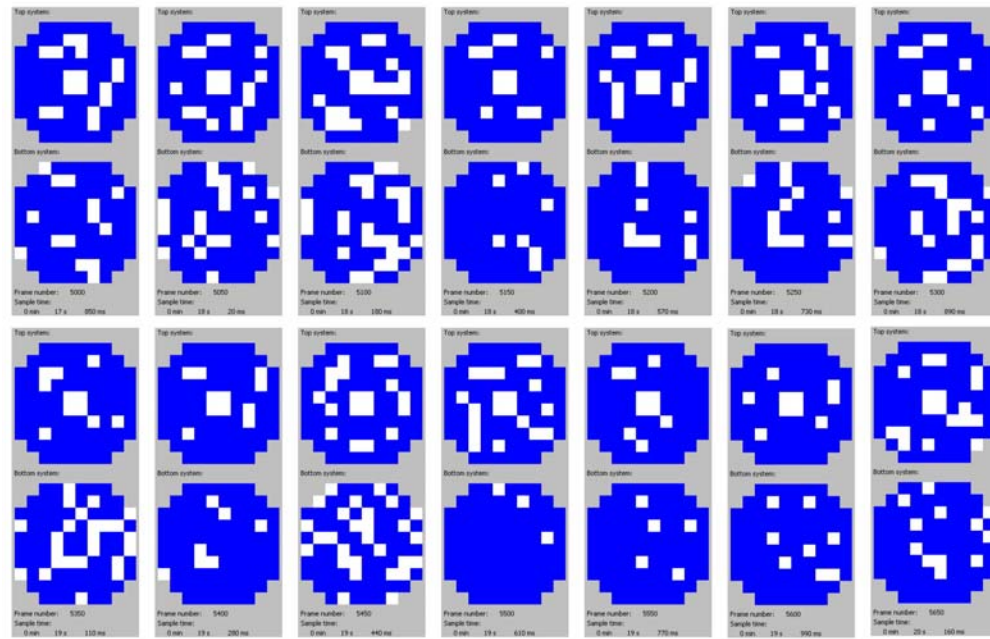
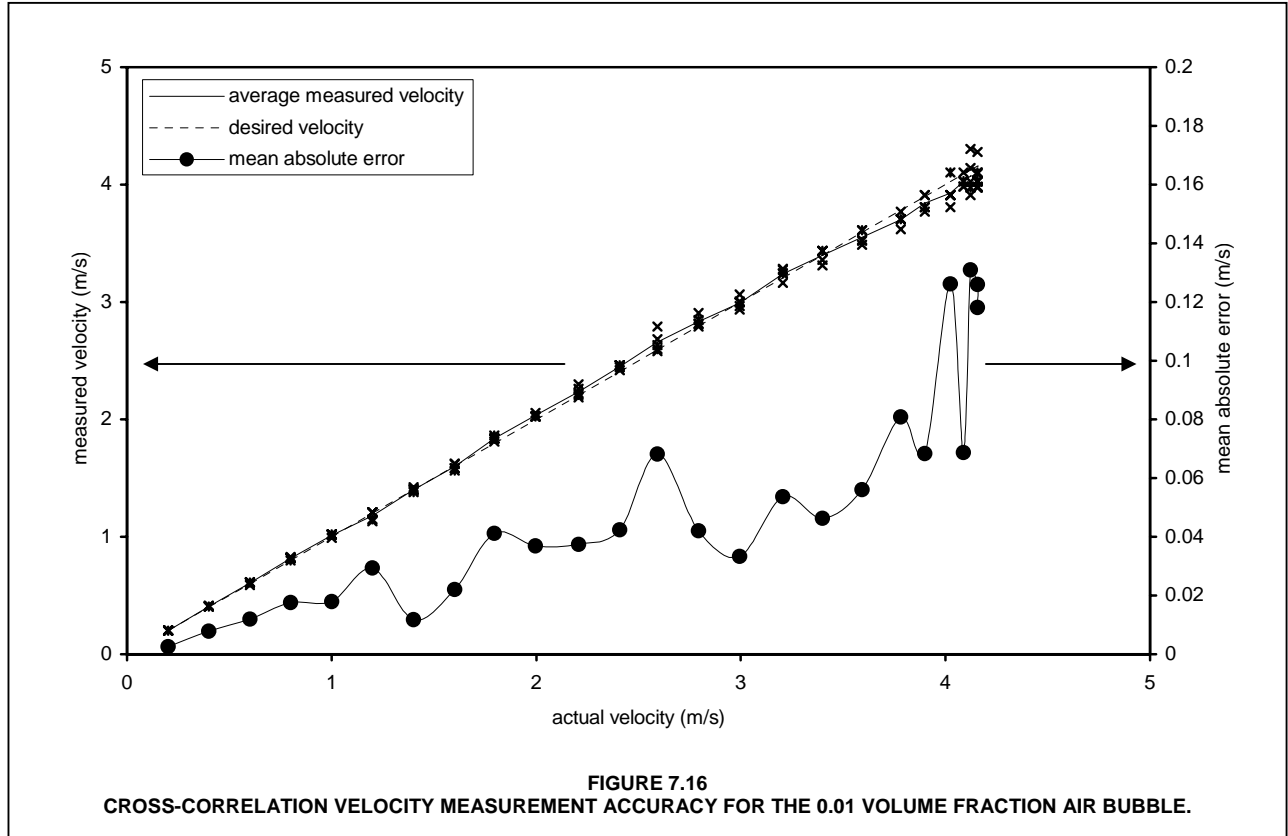


FIGURE 7.15
SCREEN CAPTURES OF A HOMOGENOUS BUBBLE FLOW WHERE, FOR EACH SCREEN CAPTURE, THE TOP FRAME IS THE TOP MEASUREMENT PLANE AND THE BOTTOM FRAME IS THE BOTTOM MEASUREMENT PLANE. THESE IMAGE RECONSTRUCTIONS WERE PERFORMED ON-LINE AND THEIR RESULTS DISPLAYED IN REAL-TIME. THE ORDER OF THE FRAMES IS LEFT TO RIGHT AND ONLY EVERY 50TH FRAME IS CAPTURED. AS BEFORE, THE BLACK IS THE WATER PHASE, THE WHITE IS THE AIR PHASE AND THE LIGHT GREY REPRESENTS THE AREA OUTSIDE THE PIPELINE.

Further, figure 7.15 shows a sequence of image reconstructions performed on-line during the test. For each screen capture, the top frame is the reconstruction of the top measurement plane and the bottom frame is the reconstruction of the bottom plane. The order of the screen captures is left to right and only every 50th frame is captured. No means was provided to verify these results other than visual inspection of the rig during a test. Subsequently, it cannot be said that the system is detecting individual bubbles, although it is certainly true that the system is detecting groups of bubbles. Although the results of this test are promising, better reconstruction results will be achieved with a neural network if some partial separation exists between the multiple phases in the pipeline under investigation. Whether this can be achieved or not depends on the physical limitations and requirements of the intended application.

7.3.4 Cross-correlation velocity prediction results

In order to assess the accuracy of the cross-correlation velocity prediction, a number of tests were conducted using the 0.01 volume fraction air bubble at different simulated velocities. A comparison was made between the simulated air phase velocity and the velocity measured by the cross-correlation algorithm and the results are plotted in figure 7.16. Included in figure 7.16 is the mean absolute error between the actual velocity and the measured velocity. As expected, the mean absolute error increases as the velocity of the bubble increases. This is a result of the reduction in the transit time between the top and bottom measurement planes. Since the sampling interval is fixed, the accuracy with which this transit time can be determined is reduced as the number of frames delay, determined by the cross-correlation algorithm, decreases. In addition, it was observed that the slippage between the drive belt and the stepper motor pulley increased as the flow velocity increased.



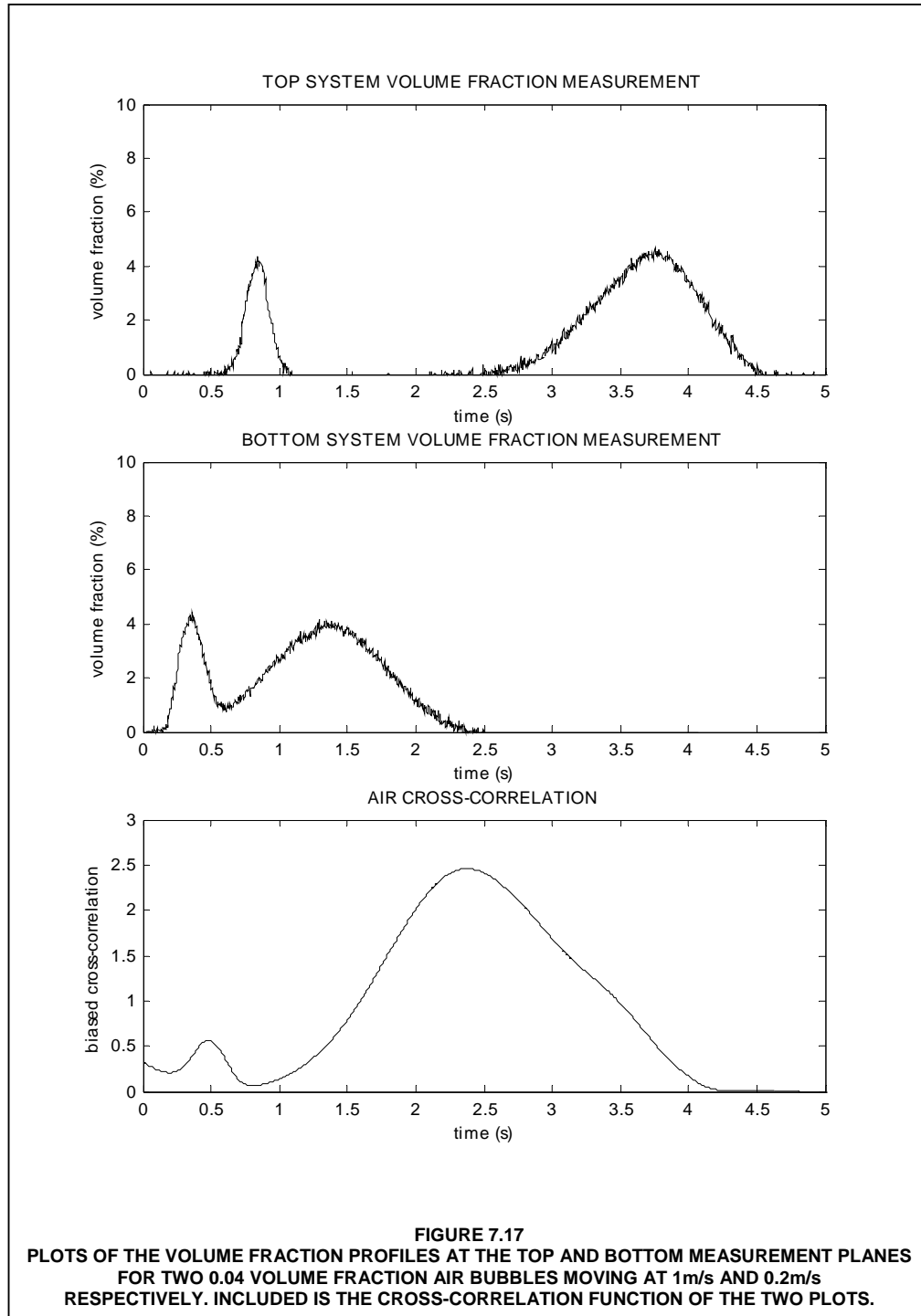
The theoretical velocity discrimination of this system can be calculated using equation 4.1 on page 18 in Chapter 4 as:

$$\frac{\Delta V}{V_{MAX}} = \frac{V_{MAX}}{2vL} \quad (7.12)$$

$$\frac{\Delta V}{V_{MAX}} = \frac{4.2 \text{ m/s}}{2(280 \text{ frames/s})(0.50 \text{ m})} = 1.5\% \quad (7.13)$$

From figure 7.16, the velocity discrimination at the maximum simulated velocity of 4.2m/s is $\frac{0.13}{4.2} = 3.1\%$, which is comparable to the theoretical limit as calculated using equation 7.12. Having determined the accuracy of the cross-correlation algorithm for the simple case of a single bubble moving through the rig, it was decided that the performance of the system should be tested on the more realistic configuration of multiple bubbles moving at different velocities. Initially, these tests were only conducted with air bubbles, although the next section will examine the results for an air and gravel bubble moving at different velocities.

Tests were conducted with multiple air bubbles moving at different velocities through the laboratory-scale rig. For example, figure 7.17 is a plot of the air volume fraction profiles at the top and bottom measurement planes for the configuration of two 0.04 volume fraction air bubbles. Included in figure 7.17 is the cross-correlation function of the volume fraction profiles. One of the bubbles was set to move at 0.2m/s, while the other bubble was set to move at 1m/s. As expected, these individual bubbles are detected at the top and bottom measurement planes, with the response corresponding to the faster bubble being narrower than the response produced by the slower bubble. In addition, it is evident that the cross-correlation function contains two peaks corresponding to the two different bubble velocities, but it is the peak resulting from the slower bubble that is detected as the average velocity. This is because of the wider response generated by the slower bubble.



However, it is not that the cross-correlation function responds only to the slower bubble but instead it responds to the bubble with the larger response. To prove this theory, the above test was repeated with a 0.01 volume fraction air bubble and a 0.04 volume fraction air bubble. The 0.01 volume fraction air bubble was set to move at 0.2m/s and the 0.04 volume fraction air bubble was set to move at 1m/s. Since the 0.04 volume fraction air bubble produced a larger response than the 0.01 volume fraction air bubble, the cross-correlation algorithm determined the average velocity to be 1m/s. This is a favourable result since it is preferable for the algorithm to respond to the large masses rather than individual bubbles in order to achieve a better assessment of the mass flow rate. The following section will verify the performance of the dual-plane 16-electrode FDM impedance tomography system on the more complex task of a three-phase air-gravel-water reconstruction.

7.4 Three-phase Air-Gravel-Water Reconstruction Results

Although the intended application requires accurate volume fraction predictions of the air, gravel and water phases, image reconstructions will be performed as a means of visually assessing the accuracy of the system. Screen captures of these image reconstruction results will be included in the following sections where appropriate.

7.4.1 Real-time volume fraction prediction and image reconstruction results

In order to perform three-phase reconstructions, the appropriate neural network reconstruction algorithms were loaded before the commencement of the real-time testing. A consequence of the increased processing required to perform a three-phase reconstruction was a reduction in the on-line frame-rate. Specifically, the frame-rate of the three-phase image reconstruction decreased to approximately 140frames/s and the frame-rate of the three-phase double-layer volume fraction predictor decreased to approximately 180frames/s. The frame-rate of the single-layer volume fraction predictor remained approximately 280frames/s. The consequences of this reduction in frame-rate will be discussed at a later stage.

Tests were conducted using both air and gravel bubbles and their dynamic volume fraction errors were recorded. Table 7.3 compares the average dynamic volume fraction errors for the different reconstruction algorithms considered at both the top and bottom measurement planes.

<p>TABLE 7.3 COMPARISON BETWEEN THE AVERAGE DYNAMIC VOLUME FRACTION ERRORS OF THE DIFFERENT NEURAL NETWORK RECONSTRUCTION TECHNIQUES CONSIDERED FOR A THREE-PHASE AIR-GRAVEL-WATER RECONSTRUCTION. ALL RESULTS REPRESENT THE ABSOLUTE ERROR AND ARE ROUNDED OFF TO TWO SIGNIFICANT DIGITS.</p>				
RECONSTRUCTION TECHNIQUE	TOP PLANE		BOTTOM PLANE	
	AIR VOLUME FRACTION ERROR (%)	GRAVEL VOLUME FRACTION ERROR (%)	AIR VOLUME FRACTION ERROR (%)	GRAVEL VOLUME FRACTION ERROR (%)
IMAGE RECONSTRUCTION	0.72	1.5	0.68	1.4
SINGLE-LAYER VOLUME FRACTION PREDICTOR	0.62	1.5	0.67	1.5
DOUBLE-LAYER VOLUME FRACTION PREDICTOR	1.0	1.5	0.89	1.6

As expected, the dynamic volume fraction error of the gravel phase is poorer than that of the air phase.

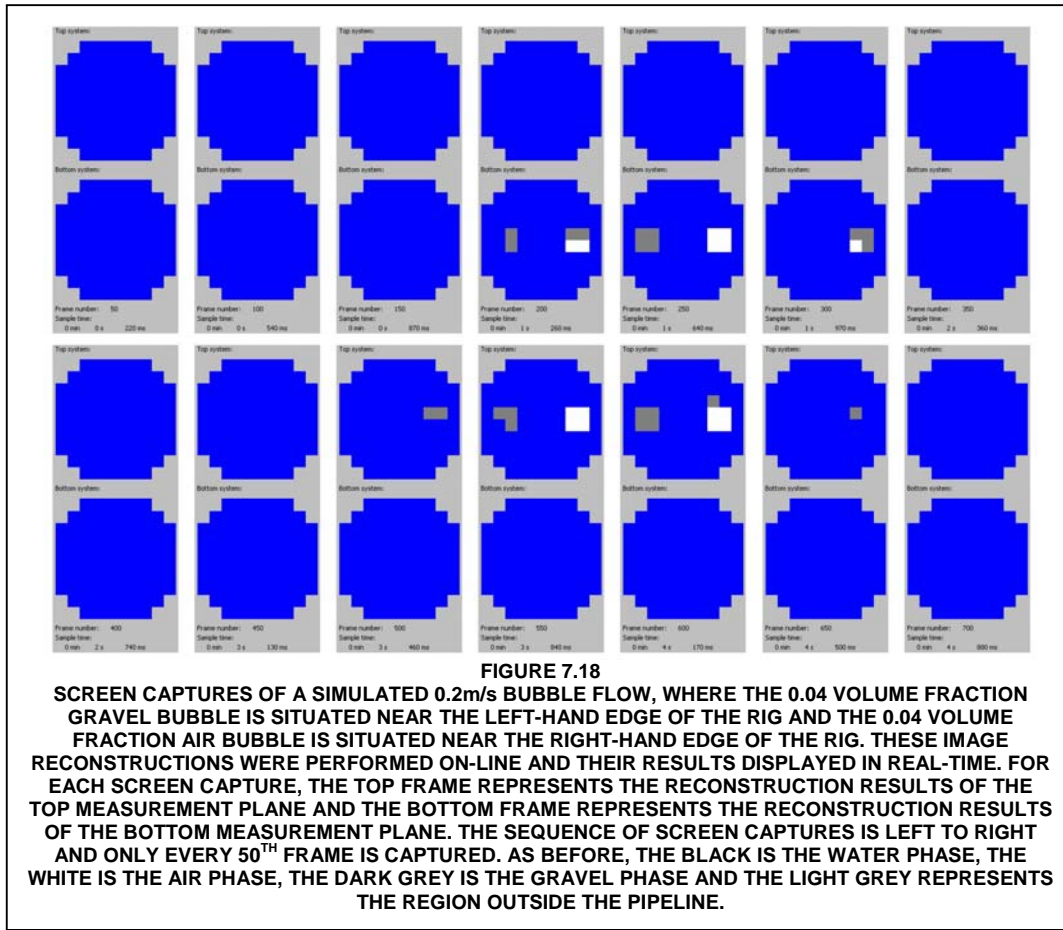


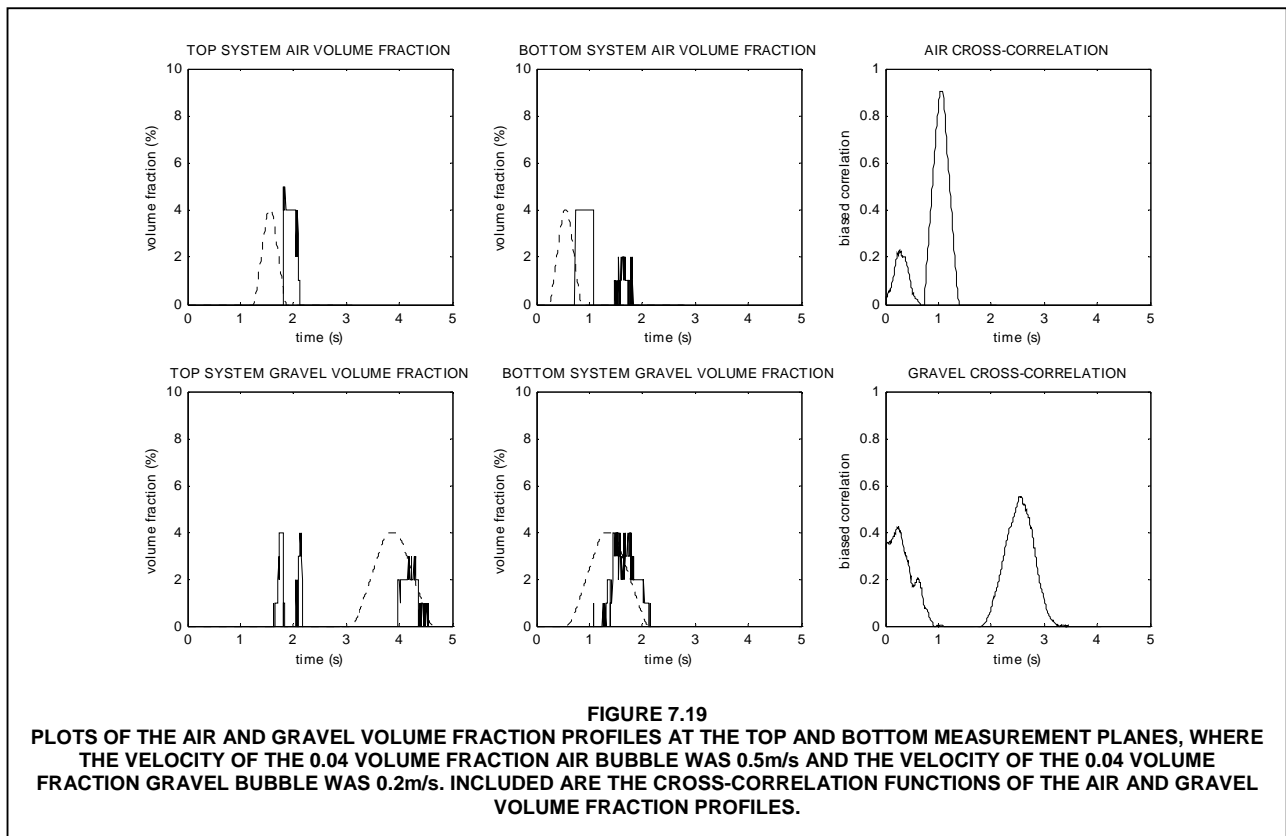
Figure 7.18 is a sequence of screen captures corresponding to the specific configuration of a 0.04 volume fraction air bubble situated near the right-hand edge of the rig and a 0.04 volume fraction gravel bubble situated near the left-hand edge of the rig. For each screen capture, the top frame represents the reconstruction results of the top measurement plane and the bottom frame represents the reconstruction results of the bottom measurement plane. The sequence of the screen captures is left to right and only every 50th frame is captured. Various characteristics common to the three-phase reconstruction results will be analysed with reference to this sequence of screen captures.

Firstly, the air bubble is detected as gravel as it enters the measurement volume. It is then correctly detected as air. As it exits the measurement volume, so the air bubble is once again detected as gravel. These gravel transition regions are similar to the two-phase air-water reconstruction results where the air bubble was first detected as a small air bubble as it entered the measurement volume. It can also be seen in figure 7.18 that no transition regions exist for the gravel bubble. In other words, the gravel bubble is correctly detected as gravel when it first enters the measurement volume. In terms of the neural network reconstruction, the changes introduced by a gravel bubble are not as defined as those introduced by an air bubble. This is because gravel has a slightly higher dielectric constant than air. Subsequently, as the air bubble enters the measurement volume, the minor changes introduced are probably comparable with those of a gravel bubble and it is detected as gravel. However, once the air bubble is completely within the measurement volume, the changes introduced are greater than those of a gravel bubble and it is correctly detected as air.

Secondly, tests revealed that the reconstruction algorithm was unable to consistently differentiate between the gravel and air phases. Although this is not evident in figure 7.18 on page 104, air bubbles would sometimes be detected only as gravel and gravel bubbles would be detected only as air. A possible explanation for this is that the capacitance readings are not accurate enough to distinguish between the air and gravel phases, since this separation is based primarily on the difference in their dielectric constants. Further, the gravel volume fraction measured by the volume fraction predictor consists of a continuous output offset. Superimposed on this offset are the changes resulting from the introduction of a bubble. This volume fraction also contains a frequency component. By performing a Fourier transform on the gravel volume fraction profile, it was determined that this ripple was at the same frequency as the ripple on the low-pass filter outputs of the synchronous detectors. This proved that the system was sensitive to this output ripple, as expected. Greater reconstruction accuracy should therefore be achieved if this output ripple were eliminated through the use of a sine wave excitation system.

7.4.2 Cross-correlation velocity prediction results

Tests were conducted to determine whether the cross-correlation algorithm was able to calculate the velocities of individual components in a multi-phase flow. These tests consisted of a 0.04 volume fraction air bubble and a 0.04 volume fraction gravel bubble moving through the laboratory-scale rig at different velocities. Figure 7.19 is an example of such a test, where the air bubble was set to move at 0.5m/s and the gravel bubble was set to move at 0.2m/s. Included in the figure are the air and gravel volume fraction profiles at the top and bottom measurement planes, as well as the air and gravel cross-correlation functions. The dashed lines represent the desired volume fraction profiles, according to the bubble size and flow-simulation apparatus configuration. An analysis of figure 7.19 reveals certain results.



Firstly, each cross-correlation function contains two peaks corresponding to the partial separation of the air and gravel bubble. In the ideal case where the reconstruction algorithm completely separates the detection of the air and gravel phases, only one peak will exist in each cross-correlation function. In this particular example, the reconstruction algorithm has achieved a satisfactory separation. Consequently, the larger peak of the air cross-correlation function correctly predicts the velocity of the air phase as 0.48m/s and the larger peak of the gravel cross-correlation function correctly predicts the velocity of the gravel phase as 0.20m/s. However, for tests where a successful separation between the two phases was not achieved, the predicted phase velocities were erroneous.

Secondly, the problem of distinguishing between the air and gravel phases is evident. Specifically, the gravel transition regions of the air bubble at the top measurement plane can be seen. Further, the gravel bubble is partly detected as an air bubble at the bottom measurement plane. Subsequently, the accuracy with which the cross-correlation algorithm calculates the velocities of the individual components will depend on the accuracy of the reconstruction algorithm to distinguish between the air and gravel phases. Although these results are limited, they do prove that a cross-correlation algorithm is able to calculate the velocities of individual components within a multi-phase flow, provided a satisfactory separation between the different phases is achieved.

7.4.3 Limitations of three-phase reconstruction algorithms

As mentioned previously, the use of three-phase reconstruction algorithms reduced the on-line frame-rate of the system. Further, the accuracy of the velocities calculated using the cross-correlation algorithm decreased and occasionally, bubbles would not be detected at certain measurement planes. This behaviour can be explained as follows: the sampling and reconstruction tasks are performed simultaneously and operate in sets of 50 frames. For the simpler two-phase reconstruction algorithms, the time taken to perform the reconstruction of 50 frames is shorter than the time taken to capture the next set of 50 frames. Hence, the computer waits for the sampling to complete and immediately instructs the capture of the next set of 50 frames once the sampling is complete. Consequently, both measurement planes are continuously sampled. In contrast, the time taken to perform the more advanced three-phase reconstruction algorithms is longer than the sampling time. Hence, the sampling system has to wait for the computer to finish the reconstruction of the previous set of 50 frames, before it starts the capture of the next set of 50 frames. A dead-time is therefore introduced in the sampling of the measurement planes. Consequently, if a bubble were to pass through a particular measurement plane during this dead-time, then it would not be detected since the system was not being sampled at that time. Further, since the frames captured do not correspond to a constant sampling interval, the transit time predicted by the cross-correlation algorithm is erroneous.

This problem can be solved through the use of a faster reconstruction computer or a reduction in the sampling rate so that the timing of the process is once again determined by the sampling and not the reconstruction.

CHAPTER 8

CONCLUSIONS

Based on the findings of this research, the following conclusions have been drawn:

1. Frequency division multiplexed impedance tomography provides a means of performing impedance tomography at a far higher frame-rate than standard time division multiplexed impedance tomography systems with minimal loss in resolution. A dual-plane 16-electrode FDM impedance tomography system has been developed that operates at an on-line real-time frame-rate of approximately 280frames/s for a two-phase image reconstruction or three-phase volume fraction prediction. Further, the resolution of the tomography system was shown to be 10% of the pipeline diameter at the centre of the pipeline, which is considered standard for electrical tomography systems.

The only major drawback of the FDM impedance tomography technique is that the conductance readings are obtained using a two-electrode collection protocol. Consequently, the conductance readings are sensitive to variations in the contact impedance between the conductance probes and the material being imaged and therefore drift considerably over time. These contact impedance variations have been shown to be primarily the result of electrochemical reactions taking place between the metal conductance probes and the seawater. In the current system, this problem was circumvented by using potable water, which has a far lower conductance than seawater. However, a feasibility study is currently being performed on implementing a frequency division multiplexed four-electrode adjacent pair measurement protocol for the conductance measurements. This would eliminate the problems associated with two-electrode conductance measurements.

2. Neural networks trained with static bubble configurations provide a means of performing accurate on-line real-time reconstructions of dynamic flows. A single-layer feed-forward neural network was trained to perform a two-phase air-water volume fraction prediction. The volume fraction error of this network for static test cases was 0.20% for the top measurement plane and 0.18% for the bottom measurement plane. For dynamic tests, the volume fraction errors increased to 0.68% for the top measurement plane and 0.58% for the bottom measurement plane. The average on-line frame-rate of this reconstruction algorithm is 280frames/s.

A double-layer feed-forward neural network was trained to perform a three-phase air-gravel-water volume fraction prediction. The volume fraction errors of this network for static test cases were 1.2%, 1.5% and 0.86% for the air, gravel and water phases at the top measurement plane respectively. The volume fraction errors for the air, gravel and water phases at the bottom measurement plane for static test cases were 1.4%, 1.7% and 1.0% respectively. For dynamic tests, the volume fraction errors measured at the top measurement plane were 1.0% and 1.5% for the air and gravel phases respectively and 0.89% and 1.6% at the bottom measurement plane respectively. The average on-line frame-rate of this reconstruction algorithm is 180frames/s. Although the initial results for the three-phase reconstructions were promising, the neural networks trained were unable to consistently differentiate between the gravel and air phases. This is believed to be a result of the limitations in the accuracy and resolution of the capacitance measurement circuitry resulting from the use of square wave excitation signals, although this assumption has not yet been verified.

Neural networks were also trained to perform an image reconstruction of the pipeline contents, although these were used primarily for verification purposes.

Various concerns existed regarding the use of neural networks in applications where a distributed multi-phase mixture is to be monitored instead of contiguous masses. Initial tests conducted on a homogenous bubble flow illustrated that the neural networks developed were capable of imaging such flows. However, due to the simplicity of the test apparatus, it was not possible to confirm whether the reconstruction results equated to the situation within the measurement volume.

The neural network reconstruction accuracy for situations involving the masking effect is improved by including examples of masking in the training database. Further, the reconstruction resolution is improved at the centre of the pipeline by not allowing any of the bubbles in the training database to come into contact with the conductance probes. The drawback is that distorted reconstruction results are achieved if one or more conductance probes are immersed in air. Whether this presents a problem or not depends on the specific application. For example, if the system were to be used on a horizontal pipeline containing a stratified flow, then the training database would have to be extended to include this possibility.

3. The cross-correlation algorithm can accurately measure the velocities of individual components in a multi-phase flow provided the reconstruction algorithm is able to distinguish between the different phases. Specifically, for a simulated air and gravel upward flow, the cross-correlation algorithm was able to measure the individual velocities of the air and gravel phases. Further, the cross-correlation algorithm was able to measure the velocity of a 0.01 volume fraction air bubble with a velocity discrimination of only 3.1% at the maximum bubble velocity of 4.2m/s, which is comparable to the theoretical limit for this particular system. It has also been shown that component volume fractions provide a computationally efficient means of calculating the average individual component velocities, without having to perform a pixel-by-pixel cross-correlation.
4. Square wave excitation can achieve good two-phase reconstruction results at a reasonable cost. However, to achieve better three-phase reconstruction results, and specifically improved air-gravel differentiation, greater accuracy of the conductance, and specifically the capacitance, measurements are required. Sine wave excitation systems provide an alternative to the square wave excitation system considered in this thesis, which should result in improved reconstruction accuracy since the output voltage readings would not contain the ripple inherent in a square wave excitation system. A sine wave excitation system is currently being developed for the 8-electrode prototype rig to assess the validity of this statement.

Synchronised transmitters were used in the current system to minimise the cross-plane interference when both systems operate simultaneously. A drawback of this approach is that the received signals for a particular plane of electrodes also contain components resulting from the transmitters of the other plane of electrodes. Subsequently, a large disturbance passing through the first plane of electrodes will also introduce a change to the received signals at the second plane of electrodes. Ideally, an FDM impedance tomography system would use different sets of frequencies at the two measurement planes so that synchronised transmitters are not required and the synchronous detectors automatically reject any interference components. Although this could not be implemented in the current square wave excitation system, the sine wave system being developed has made provision for the generation of different sets of frequencies at the two measurement planes.

Based on these conclusions, it can be seen that good results have been achieved using the dual-plane 16-electrode FDM impedance tomography system. A literature study revealed that this is, to the best of the author's knowledge, the first ever implementation of frequency division multiplexing to perform impedance tomography. Further, it has been shown that this technique provides a means of performing on-line real-time reconstructions at a far higher frame-rate than standard time division multiplexed impedance tomography systems with minimal loss in resolution for a given number of electrodes. Subsequently, it enables tomography to be used in industrial applications where previously the limited frame-rates of tomography systems were a hindrance, such as on-line multi-phase flowmeters. Although the frame-rate of the current system is only 280frames/s, this measurement protocol has the potential to achieve frame-rates of the order of thousands of frames per second, simply by using sine wave excitation and high bandwidth electronic components. Whether such a high frame-rate is useful or not obviously depends on the intended application. However, DebTech have identified the potential of such a system and have since obtained a provisional patent in order to assess its marketability. Further, this system is comparatively cheap to implement compared to current commercially available systems. Chapter 9 gives recommendations for future development.

CHAPTER 9

RECOMMENDATIONS FOR FUTURE DEVELOPMENT

Based on the findings and conclusions of this research, the following recommendations for future development can be made:

1. An 8-electrode FDM impedance tomography system using sine wave excitation must be developed. Tests should be conducted to assess whether an improvement in reconstruction accuracy and air-gravel separation is achieved through the use of sine wave excitation. If improved results are achieved, then this system must be extended for use on the dual-plane 16-electrode laboratory-scale rig. The various dynamic tests considered in this thesis can then be repeated and the results compared.
2. Research must be done into the drift of the conductance and capacitance readings resulting from the electrochemical reactions taking place within the rig. These tests can be performed for the simpler configuration of a 2-electrode system consisting of a single transmitter and single receiver electrode situated diametrically opposite each other. The University of Cape Town Control and Instrumentation department possess a 50mm 0-1.6m/s flow rig that can be used to conduct these tests. A special pipeline section must be constructed with these two electrodes. Initially, one end of the section can be sealed and placed vertically so that water is retained. A drift analysis should be performed for this configuration in order to provide a reference point corresponding to static water. The end seal can then be removed and the pipeline section placed within the flow loop of the flow rig. The above drift analysis should then be repeated and a comparison made. Various electrode materials could also be tested using this approach.
3. The applicability of the neural network reconstruction algorithms to the intended application should be assessed. Various modifications to the generation of the neural network training database have been proposed throughout this thesis, which should improve the overall performance of the system. These include extending the database to incorporate bubbles that are shorter than the electrode length and, depending on the intended application, incorporating an all-air training data point. Although not discussed in this thesis, it is also possible that improved air-gravel separation could be achieved by training separate neural networks to predict the air and gravel phases individually. In this way, each neural network can be optimised for the detection of a particular phase. Currently, a single neural network is trained to perform both the air and gravel volume fraction predictions. If an improvement in the air-gravel separation were not achieved through these proposed modifications, then it would be advisable to re-examine the positioning of the tomography system in the intended application. Specifically, greater accuracy would be obtained if some form of partial separation of the phases in the flow had already been achieved. In terms of the current application, it has been proposed that the impedance tomography system is positioned before the introduction of the air phase in the airlift, thus simplifying the measurement task to that of a two-phase gravel-seawater reconstruction.

These tests should be conducted before any decision is made regarding whether the project should continue. It should be noted that industrialising such an instrument would not be a trivial task. In particular, certain challenges would need to be overcome, such as the selection of the pipeline material and the conductance probe material for the harsh environment of an offshore airlift. In addition, further testing should be done to assess how the system would perform on real flows that simulate the dynamics of the intended application more realistically. Further recommendations, specific to this particular system, are discussed in Chapter 10.

CHAPTER 10

REMAINING SYSTEM ISSUES THAT STILL NEED TO BE ADDRESSED

The following is a brief discussion of certain issues that must still be addressed in this particular system before further research is conducted.

The current system is based on the Eagle PC30G data acquisition card. This card is intended for an ISA slot and is soon to be obsolete since it is increasingly difficult to purchase computer motherboards that support the ISA standard. It is recommended that a PCI-based National Instruments card be purchased for future work. The sampling software would need to be modified to use this new card. However, since the device-specific calling procedures are encapsulated in a sampling class, it would only be necessary to modify this sampling class. The current data capture rate of the Eagle PC30G data acquisition card is sufficient.

There is an occasional loss of synchronisation between the sample controller board and the data acquisition card. This loss of synchronisation is the result of false triggering of the data acquisition card by noise on the external trigger line and the effects are temporary. Replacing the ribbon cable connecting the sample controller board to the data acquisition card with a multi-core shielded cable should solve this problem. In addition, the electronics should be modified to improve the noise separation between the digital and analogue ground planes.

The current flow-simulation apparatus is extremely limited in terms of the test cases that it is able to simulate. Slippage between the drive belt and pulley means that it is impossible to predict when exactly a bubble should be passing through a measurement section. Consequently, it cannot be used to verify the time delay response of the measurement electronics. In addition, it is limited in terms of the size of the bubble and the maximum velocity that it can simulate. The o-ring drive belt and pulley arrangement should be replaced by a toothed-belt and sprocket arrangement. Further, higher torque stepper motors should be used.

The MOSFET drivers in the transmitters should be replaced with break-before-make MOSFET drivers. Although the cross-conduction circuitry discussed previously greatly reduced the cross-conduction in the push-pull power MOSFET output stage, the output square waves produced are still not ideal and contain minor irregularities. In addition, switching noise is still evident on the power supply lines.

LIST OF REFERENCES

1. Smit Q 2000 Material phase detection using capacitance tomography *MSc. Thesis* University of Cape Town
2. Smit Q, Tapson J and Mortimer B J P 2000 Material phase detection system using capacitance tomography *Proc. IEEE-IMTC Conf. (Baltimore, WA)*
3. Teague G 2000 Neural network reconstruction for electrical capacitance tomography system *BSc. Thesis* University of Cape Town
4. Teague G, Tapson J and Smit Q 2001 Neural network reconstruction for tomography of a gravel-air-seawater mixture *Meas. Sci. Technol.* **12** 1102-1108
5. Jaworski A J and Dyakowski T 2001 Application of electrical capacitance tomography for measurement of gas-solids flow characteristics in a pneumatic conveying system *Meas. Sci. Technol.* **12** 1109-1119
6. Gamio J C, Yang W Q and Stott A L 2001 Analysis of non-ideal characteristics of an ac-based capacitance transducer for tomography *Meas. Sci. Technol.* **12** 1076-1082
7. Reinecke N and Mewes D 1996 Recent developments and industrial/research applications of capacitance tomography *Meas. Sci. Technol.* **7** 233-246
8. Basarab-Horwath I, Daniels A T and Green R G 2001 Image analysis in dual modality tomography for material classification *Meas. Sci. Technol.* **12** 1153-1156
9. Jaworski A J and Dyakowski T 2001 Tomographic Measurements of Solids Mass Flow in Dense Pneumatic Conveying. What Do We Need to Know About the Flow Physics? *Proc. 2nd World Congr. on Industrial Process Tomography (Hannover)* pp 353-361
10. Loh W W, Waterfall R C, Cory J and Lucas G P 1999 Using ERT for Multi-Phase Flow Monitoring *Proc. 1st World Congr. on Industrial Process Tomography (Buxton)* pp 47-53
11. Byars M 2001 Developments in Electrical Capacitance Tomography *Proc. 2nd World Congr. on Industrial Process Tomography (Hannover)* pp 542-549
12. Deng X, Dong F, Xu L J, Liu X P and Xu L A 2001 The design of a dual-plane ERT system for cross correlation measurement of bubbly gas/liquid pipe flow *Meas. Sci. Technol.* **12** 1024-1031
13. Thorn R, Johansen G A and Hammer E A 1999 Three-Phase Flow Measurement in the Offshore Oil Industry Is There a Place for Process Tomography? *Proc. 1st World Congr. on Industrial Process Tomography (Buxton)* pp 228-235
14. Arko A, Waterfall R C and Beck M S 1999 Development of Electrical Capacitance Tomography for Solids Mass Flow Measurement and Control of Pneumatic Conveying Systems *Proc. 1st World Congr. on Industrial Process Tomography (Buxton)* pp 140-146
15. Dickin F J *et al* 1992 Tomographic imaging of industrial process equipment: techniques and applications *Proc. IEE* **139** 72-81
16. Yang W Q 2001 Advance in AC-based capacitance tomography system *Proc. 2nd World Congr. on Industrial Process Tomography (Hannover)* pp 557-564
17. Dickin F J, Williams R A and Beck M S 1993 Determination of composition and motion of multicomponent mixtures in process vessels using electrical impedance tomography – I. Principles and process engineering applications *Chem. Eng. Sci.* **48** 1883-1897
18. Yuen E L, Mann R, York T A and Grieve B D 2001 Electrical Resistance Tomography (ERT) Imaging of a Metal-Walled Solid-Liquid Filter *Proc. 2nd World Congr. on Industrial Process Tomography (Hannover)* pp 183-190

19. Cilliers J J, Xie W, Neethling S J, Randall E W and Wilkinson A J 2001 Electrical resistance tomography using a bi-directional current pulse technique *Meas. Sci. Technol.* **12** 997-1001
20. van Weereld J J A, Collie D A L and Player M A 2001 A fast resistance measurement system for impedance tomography using a bipolar DC pulse method *Meas. Sci. Technol.* **12** 1002-1011
21. Hoyle B S, Bailey N J and Nooralahiyan A Y 1995 Performance of neural network in capacitance-based tomographic process measurement systems *Meas. Control* **28** 109-112
22. West R M, Tapp H S, Spink D M, Bennett M A and Williams R A 2001 Application-specific optimization of regularization for electrical impedance tomography *Meas. Sci. Technol.* **12** 1050-1054
23. Ge S M and Yang W Q 2001 Filtering in Image Reconstruction for Electrical Capacitance Tomography *Proc. 2nd World Congr. on Industrial Process Tomography (Hannover)* pp 41-47
24. Sun T D, Mudde R, Schouten J C, Scarlett B and van den Bleek C M 1999 Image Reconstruction of an Electrical Capacitance Tomography System Using an Artificial Neural Network *Proc. 1st World Congr. on Industrial Process Tomography (Buxton)* pp 174-180
25. West R M, Jia X and Williams R A 1999 Parametric Modelling in Industrial Process Tomography *Proc. 1st World Congr. on Industrial Process Tomography (Buxton)* pp 444-450
26. Isaksen Ø 1996 A review of reconstruction techniques for capacitance tomography *Meas. Sci. Technol.* **7** 325-337
27. Dickin F and Wang M 1996 Electrical resistance tomography for process applications *Meas. Sci. Technol.* **7** 247-260
28. Loh W W and Dickin F J 1996 Improved modified Newton-Raphson algorithm for electrical impedance tomography *Electron. Lett.* **32** 206-207
29. Bishop C M 1995 *Neural Networks for Pattern Recognition* (New York: Oxford University Press)
30. Nooralahiyan A Y, Hoyle B S and Bailey N J 1994 Neural network for pattern association in electrical capacitance tomography *Proc. IEE* **141** 517-521
31. Nooralahiyan A Y and Hoyle B S 1997 Three-component tomographic flow imaging using artificial neural network reconstruction *Chem. Eng. Sci.* **52** 2139-2148
32. Ratajewicz-Mikolajczak E, Shirkoohi G H and Sikora J 1998 Two ANN Reconstruction Methods for Electrical Impedance Tomography *IEEE Trans. Magn.* **34** 2964-2967
33. Adler A and Guardo R 1994 A Neural Network Image Reconstruction Technique for Electrical Impedance Tomography *IEEE Trans. Med. Imag.* **13** 594-560
34. Riedmiller M and Braun H 1993 A Direct Adaptive Method for Faster Backpropagation Learning: The RPROP Algorithm *Proc. IEEE Int. Conf. on Neural Networks (ICNN) (San Francisco)* ed H Ruspini, pp 586-591
35. Riedmiller M 1994 Advanced Supervised Learning in Multi-layer Perceptrons – From Backpropagation to Adaptive Learning Algorithms *Int. J. of Computers Standards and Interfaces* **5**
36. Riedmiller M 1994 Rprop – Description and Implementation Details *Technical Report* University of Karlsruhe
37. Szczepanik Z and Rucki Z 2001 Problems of the Data Acquisition Rate for Impedance Tomography *Proc. 2nd World Cong. on Industrial Process Tomography (Hannover)* pp 511-516
38. Horowitz P and Hill W 1995 *The art of electronics* (Cambridge: Cambridge University Press)
39. Georgakopoulos D, Waterfall R C and Yang W Q 2001 Towards the Development of a Multiple-Frequency ECT System *Proc. 2nd World Cong. on Industrial Process Tomography (Hannover)* pp 550-556
40. Brown B H 2001 Medical impedance tomography and process impedance tomography: a brief review *Meas. Sci. Technol.* **12** 991-996
41. Morrison N 1994 *Introduction to Fourier Analysis* (Toronto: John Wiley and Sons)

42. Hervieu E and Seleglim P Jr 1999 Direct Imaging of Two-Phase Flows by Electrical Impedance Measurements *Proc. 1st World Cong. on Industrial Process Tomography (Buxton)* pp 62-69
43. Deng X, Dong F, Xu L J, Liu X P and Xu L A 2001 Measurement of the Gas Phase Velocity in Gas-Liquid Flows Using a Dual-Plane ERT System *Proc. 2nd World Cong. on Industrial Process Tomography (Hannover)* pp 669-676
44. Georgakopoulos D and Yang W Q 2001 Key Issues in the Design of a Capacitance Transducer for Tomography *Proc. 2nd World Cong. on Industrial Process Tomography (Hannover)* pp 586-594
45. Ramphal V 2002 Implementation of MOSFETs in an impedance tomography system *BTech. thesis* Cape Technikon
46. Yang W Q 1996 Calibration of capacitance tomography systems: a new method for setting system measurement range *Meas. Sci. Technol.* **7** 863-867
47. Xie C G *et al* 1992 Electrical capacitance tomography for flow imaging: system model for development of image reconstruction algorithms and design of primary sensors *Proc. IEE* **139** 89-98
48. Xie C G, Stott A L, Plaskowski A and Beck M S 1990 Design of capacitance electrodes for concentration measurement of two-phase flow *Meas. Sci. Technol.* **1** 65-78
49. Loh W W, Dickin F J, Waterfall R C and Pinheiro P A T 1998 Alternative inter-plane data collection protocol for electrical resistance tomography in flow monitoring applications *Electron. Lett.* **34** 1487-1488
50. Beck M S and Plaskowski A 1987 *Cross Correlation Flowmeters – their Design and Application* (Bristol: Adam Hilger)
51. Yan H, Shao F and Wang S 1999 Simulation Study of Capacitance Tomography Sensors *Proc. 1st World Cong. on Industrial Process Tomography (Buxton)* pp 388-394
52. Xu H, Yang G and Wang S 1999 Effect of Axial Guard Electrodes on Sensing Field of Capacitance Tomographic Sensor *Proc. 1st World Cong. on Industrial Process Tomography (Buxton)* pp 348-352
53. Wang M 1999 Three-dimensional Effects in Electrical Impedance Tomography *Proc. 1st World Cong. on Industrial Process Tomography (Buxton)* pp 410-415
54. Ma Y, Wang H, Xu L A and Jiang C 1997 Simulation study of the electrode array used in an ERT system *Chem. Eng. Sci.* **52** 2197-2203
55. Ma Y, Xu L A and Jiang C 1999 Experimental Study of the Guard Electrodes in an ERT System *Proc. 1st World Cong. on Industrial Process Tomography (Buxton)* pp 335-338
56. Yang W Q and York T A 1999 New AC-based capacitance tomography system *Proc. IEE* **146** 47-53
57. Lucas G P, Cory J C and Waterfall R C 2000 A six-electrode local probe for measuring solids velocity and volume fraction profiles in solids-water flows *Meas. Sci. Technol.* **11** 1498-1509
58. Pinheiro P A T, Loh W W and Dickin F J 1998 Optimal sized electrodes for electrical resistance tomography *Electron. Lett.* **34** 69
59. Dyakowski T 1996 Process tomography applied to multi-phase flow measurement *Meas. Sci. Technol.* **7** 343-353
60. National Semiconductor 1999 *LMF100 High Performance Dual Switched Capacitor Filter datasheet*
61. Baluja S 1994 *Population-Based Incremental Learning: A Method for Integrating Genetic Search Based Function Optimization and Competitive Learning* Carnegie Mellon University
62. Giannopoulos A 2003 Investigation of sine-wave inputs for an FDM EIT system *MSc. Thesis* University of Cape Town
63. Eagle Technology 1999 EDR Software developers kit for Eagle Technology boards *User Manual For Dos and Windows*

64. Bendat J S and Piersol A G 1980 *Engineering applications of correlation and spectral analysis* (Toronto: John Wiley and Sons)
65. Neuffer D, Alvarez A, Owens D H, Ostrowski K L, Luke S P and Williams R A 1999 Control of Pneumatic Conveying using ECT *Proc. 1st World Cong. on Industrial Process Tomography (Buxton)* pp 71-76
66. Fransolet E, Crine M, L'Homme G, Toye D and Marchot P 2001 Analysis of electrical resistance tomography measurements obtained on a bubble column *Meas. Sci. Technol.* **12** 1055-1060
67. Bennett M A, Luke S P, Jia X, West R M and Williams R A 1999 Analysis and Flow Regime Identification of Bubble Column Dynamics *Proc. 1st World Cong. on Industrial Process Tomography (Buxton)* pp 54-61

APPENDIX A
TABLE OF THE MANUFACTURING COSTS FOR 16-ELECTRODE
FREQUENCY DIVISION MULTIPLEXED IMPEDANCE TOMOGRAPHY
SYSTEM

DESCRIPTION	APPROXIMATE COST IN US DOLLARS
COMPUTER EQUIPMENT	\$286
PRINTED CIRCUIT BOARD MANUFACTURE AND DESIGN	\$1720
ELECTRONIC COMPONENTS	\$1220
MANUFACTURE OF LABORATORY-SCALE RIG	\$2727
OTHER MATERIAL COSTS	\$148

APPENDIX B
CIRCUIT DIAGRAMS FOR 8-ELECTRODE FREQUENCY DIVISION
MULTIPLEXED IMPEDANCE TOMOGRAPHY SYSTEM

CIRCUIT DIAGRAM OF TRANSMITTER, CAPACITANCE RECEIVER AND CONDUCTANCE RECEIVER

CIRCUIT DIAGRAM SHOWING PIC16F84 FREQUENCY GENERATOR AND INPUT STAGE CONSISTING OF INVERTERS AND BUFFERS FOR THE SYNCHRONOUS DETECTION OF THE RECEIVED SIGNALS

**CIRCUIT DIAGRAM SHOWING SYNCHRONOUS DETECTION OF THE CAPACITANCE AND CONDUCTANCE
SIGNALS FROM RECEIVER A AND RECEIVER B**

**CIRCUIT DIAGRAM SHOWING SYNCHRONOUS DETECTION OF THE CAPACITANCE AND CONDUCTANCE
SIGNALS FROM RECEIVER C AND RECEIVER D**

**CIRCUIT DIAGRAM OF MULTIPLEXING STAGE FOR CAPTURE OF THE 16 CAPACITANCE AND
16 CONDUCTANCE READINGS USING A PC30G DATA ACQUISITION CARD**

APPENDIX C
CIRCUIT DIAGRAMS FOR 16-ELECTRODE FREQUENCY DIVISION
MULTIPLEXED IMPEDANCE TOMOGRAPHY SYSTEM

**CIRCUIT DIAGRAM OF THE DIFFERENT TRANSMITTERS, WHERE THE HIGH CURRENT TRANSMITTER
HAS AN ADDITIONAL PUSH-PULL POWER MOSFET OUTPUT STAGE**

**CIRCUIT DIAGRAM OF CAPACITANCE AND CONDUCTANCE RECEIVER SHOWING ADDITIONAL
LOW-PASS AND BANDPASS FILTER STAGES**

CIRCUIT DIAGRAM OF FREQUENCY GENERATOR BOARD

**CIRCUIT DIAGRAM OF INPUT STAGE AND MULTIPLEXING OF THE RECEIVER A AND B
SYNCHRONOUS DETECTOR BOARD**

**CIRCUIT DIAGRAM SHOWING THE SYNCHRONOUS DETECTION OF THE CAPACITANCE AND CONDUCTANCE
SIGNAL FROM RECEIVER A**

**CIRCUIT DIAGRAM SHOWING THE SYNCHRONOUS DETECTION OF THE CAPACITANCE AND CONDUCTANCE
SIGNAL FROM RECEIVER B**

CIRCUIT DIAGRAM OF THE SAMPLE CONTROLLER BOARD

APPENDIX D

TABLES OF THE CAPACITANCE AND CONDUCTANCE READINGS FOR 16-ELECTRODE FREQUENCY DIVISION MULTIPLEXED IMPEDANCE TOMOGRAPHY SYSTEM

RESISTANCE READINGS OF THE DIFFERENT TRANSMITTER-RECEIVER ELECTRODE PAIRS AT THE BOTTOM MEASUREMENT PLANE FOR THE LABORATORY-SCALE RIG FILLED COMPLETELY WITH POTABLE WATER

	RxA	RxB	RxC	RxD	RxE	RxF	RxG	RxH
TxA	3.45k	3.38k	3.41k	3.45k	3.43k	3.48k	3.40k	3.61k
TxB	3.47k	3.26k	3.16k	3.32k	3.31k	3.37k	3.31k	3.52k
TxC	3.62k	3.53k	3.27k	3.28k	3.39k	3.47k	3.42k	3.66k
TxD	3.62k	3.54k	3.40k	3.27k	3.23k	3.43k	3.39k	3.64k
TxE	3.71k	3.64k	3.52k	3.50k	3.32k	3.37k	3.44k	3.72k
TxF	3.74k	3.68k	3.57k	3.57k	3.51k	3.41k	3.34k	3.73k
TxG	3.61k	3.58k	3.48k	3.49k	3.44k	3.46k	3.25k	3.49k
TxH	3.55k	3.63k	3.54k	3.57k	3.53k	3.57k	3.47k	3.57k

CAPACITANCE READINGS OF THE DIFFERENT TRANSMITTER-RECEIVER ELECTRODE PAIRS AT THE BOTTOM MEASUREMENT PLANE FOR THE LABORATORY-SCALE RIG FILLED COMPLETELY WITH POTABLE WATER

	RxA	RxB	RxC	RxD	RxE	RxF	RxG	RxH
TxA	40.3p	39.9p	36.8p	33.9p	32.9p	35.5p	35.7p	41.6p
TxB	43.7p	42.0p	37.7p	34.1p	32.9p	33.4p	35.2p	41.4p
TxC	43.0p	39.6p	37.7p	34.3p	33.2p	37.0p	38.0p	41.2p
TxD	44.6p	46.4p	42.0p	40.7p	40.8p	35.5p	37.3p	42.5p
TxE	42.2p	43.3p	42.0p	38.3p	36.0p	40.1p	37.5p	41.0p
TxF	42.4p	39.0p	38.5p	32.0p	33.5p	39.4p	42.7p	42.6p
TxG	39.1p	40.1p	34.4p	32.3p	32.6p	34.0p	38.0p	42.9p
TxH	41.6p	36.0p	35.7p	33.4p	33.0p	37.5p	36.1p	42.5p

CAPACITANCE READINGS OF THE DIFFERENT TRANSMITTER-RECEIVER ELECTRODE PAIRS AT THE BOTTOM MEASUREMENT PLANE FOR THE LABORATORY-SCALE RIG FILLED COMPLETELY WITH AIR

	RxA	RxB	RxC	RxD	RxE	RxF	RxG	RxH
TxA	9.7p	9.2p	7.6p	6.8p	4.9p	5.2p	6.5p	7.2p
TxB	6.3p	8.5p	7.6p	6.3p	7.6p	5.9p	6.2p	6.7p
TxC	3.6p	3.6p	4.0p	6.3p	5.4p	5.9p	6.3p	5.7p
TxD	3.5p	3.4p	3.2p	4.1p	5.3p	4.5p	4.7p	4.4p
TxE	3.1p	3.0p	2.9p	2.8p	4.1p	6.3p	5.6p	4.8p
TxF	5.0p	4.9p	4.8p	4.5p	4.6p	6.2p	8.9p	8.3p
TxG	5.1p	4.8p	4.9p	4.4p	5.1p	5.9p	12.0p	11.6p
TxH	7.2p	5.0p	4.6p	3.0p	2.9p	3.8p	4.1p	4.1p

APPENDIX E

MICROCONTROLLER PROGRAM CODE FOR 8-ELECTRODE FREQUENCY DIVISION MULTIPLEXED IMPEDANCE TOMOGRAPHY SYSTEM

INCLUDE <PIC1684.SRC>	MOVWF PORTB	NOP	MOVWF PORTB
START BSF STATUS, RP0	NOP	MOVLW B'10011100'	NOP
;select page 1	NOP	MOVWF PORTB	NOP
CLRF TRISB	MOVLW B'00100100'	NOP	MOVLW B'11101110'
;set PORTB to all	MOVWF PORTB	NOP	MOVWF PORTB
;outputs	NOP	MOVLW B'10010100'	NOP
BCF STATUS, RP0	NOP	MOVWF PORTB	NOP
;select page 0	MOVLW B'00100100'	NOP	MOVLW B'01111110'
CLRF PORTB	MOVWF PORTB	NOP	MOVWF PORTB
;clear all outputs on	NOP	MOVLW B'11000110'	NOP
;PORTB	NOP	MOVWF PORTB	NOP
	MOVLW B'10110100'	NOP	MOVLW B'01110110'
	MOVWF PORTB	NOP	MOVWF PORTB
LOOP MOVLW B'00000000'	NOP	MOVLW B'11000110'	NOP
MOVWF PORTB	NOP	MOVWF PORTB	NOP
NOP	MOVLW B'10110100'	NOP	MOVLW B'00110111'
NOP	MOVWF PORTB	NOP	MOVWF PORTB
MOVLW B'00000000'	NOP	MOVLW B'01000110'	NOP
MOVWF PORTB	NOP	MOVWF PORTB	NOP
NOP	MOVLW B'11110010'	NOP	MOVLW B'00010111'
NOP	MOVWF PORTB	NOP	MOVWF PORTB
MOVLW B'10000000'	NOP	MOVLW B'01100110'	NOP
MOVWF PORTB	NOP	MOVWF PORTB	NOP
NOP	MOVLW B'11010010'	NOP	MOVLW B'10010111'
NOP	MOVWF PORTB	NOP	MOVWF PORTB
MOVLW B'10100000'	NOP	MOVLW B'00100011'	NOP
MOVWF PORTB	NOP	MOVWF PORTB	NOP
NOP	MOVLW B'01010010'	NOP	MOVLW B'10010111'
NOP	MOVWF PORTB	NOP	MOVWF PORTB
MOVLW B'11100010'	NOP	MOVLW B'00100011'	NOP
MOVWF PORTB	NOP	MOVWF PORTB	NOP
NOP	MOVLW B'01010010'	NOP	MOVLW B'11000001'
NOP	MOVWF PORTB	NOP	MOVWF PORTB
MOVLW B'11101010'	NOP	MOVLW B'10110011'	NOP
MOVWF PORTB	NOP	MOVWF PORTB	NOP
NOP	MOVLW B'00000011'	NOP	MOVLW B'11000001'
NOP	MOVWF PORTB	NOP	MOVWF PORTB
MOVLW B'01111010'	NOP	MOVLW B'10110011'	NOP
MOVWF PORTB	NOP	MOVWF PORTB	NOP
NOP	MOVLW B'00001011'	NOP	MOVLW B'01000001'
NOP	MOVWF PORTB	NOP	MOVWF PORTB
MOVLW B'01111010'	NOP	MOVLW B'11110001'	NOP
MOVWF PORTB	NOP	MOVWF PORTB	NOP
NOP	MOVLW B'10001011'	NOP	MOVLW B'01100001'
NOP	MOVWF PORTB	NOP	MOVWF PORTB
MOVLW B'00111011'	NOP	MOVLW B'11011001'	NOP
MOVWF PORTB	NOP	MOVWF PORTB	NOP
NOP	MOVLW B'10101011'	NOP	MOVLW B'00100000'
NOP	MOVWF PORTB	NOP	MOVWF PORTB
MOVLW B'00011011'	NOP	MOVLW B'01011001'	NOP
MOVWF PORTB	NOP	MOVWF PORTB	NOP
NOP	MOVLW B'11101001'	NOP	MOVLW B'00101000'
NOP	MOVWF PORTB	NOP	MOVWF PORTB
MOVLW B'10011111'	NOP	MOVLW B'01011001'	NOP
MOVWF PORTB	NOP	MOVWF PORTB	NOP
NOP	MOVLW B'11101001'	NOP	MOVLW B'10111000'
NOP	MOVWF PORTB	NOP	MOVWF PORTB
MOVLW B'10011111'	NOP	MOVLW B'00001000'	NOP
MOVWF PORTB	NOP	MOVWF PORTB	NOP
NOP	MOVLW B'01111101'	NOP	MOVLW B'10111000'
NOP	MOVWF PORTB	NOP	MOVWF PORTB
MOVLW B'11001101'	NOP	MOVLW B'00001000'	NOP
MOVWF PORTB	NOP	MOVWF PORTB	NOP
NOP	MOVLW B'01111101'	NOP	MOVLW B'11111010'
NOP	MOVWF PORTB	NOP	MOVWF PORTB
MOVLW B'11001101'	NOP	MOVLW B'10001100'	NOP
MOVWF PORTB	NOP	MOVWF PORTB	NOP
NOP	MOVLW B'00111100'	NOP	MOVLW B'11011010'
NOP	MOVWF PORTB	NOP	MOVWF PORTB
MOVLW B'01001101'	NOP	MOVLW B'10101100'	NOP
MOVWF PORTB	NOP	MOVWF PORTB	NOP
NOP	MOVLW B'00011100'	NOP	MOVLW B'01011110'
NOP	MOVWF PORTB	NOP	MOVWF PORTB
MOVLW B'01100101'	NOP	MOVLW B'11101110'	NOP
MOVWF PORTB	NOP		

NOP	MOVLW B'10111111'	MOVWF PORTB	NOP
MOVLW B'01011110'	MOVWF PORTB	NOP	NOP
MOVWF PORTB	NOP	MOVLW B'00101100'	MOVWF PORTB
NOP	MOVLW B'11111101'	MOVWF PORTB	NOP
MOVLW B'00001111'	MOVWF PORTB	NOP	NOP
MOVWF PORTB	NOP	MOVLW B'00101100'	MOVWF PORTB
NOP	MOVLW B'11011101'	MOVWF PORTB	NOP
NOP	MOVWF PORTB	NOP	NOP
MOVLW B'00001111'	NOP	MOVLW B'10111100'	MOVWF PORTB
MOVWF PORTB	NOP	MOVWF PORTB	NOP
NOP	MOVLW B'01011101'	NOP	NOP
NOP	MOVWF PORTB	MOVLW B'10110100'	MOVWF PORTB
MOVLW B'10001111'	NOP	MOVWF PORTB	NOP
MOVWF PORTB	NOP	NOP	NOP
NOP	MOVLW B'01010101'	MOVLW B'11110110'	MOVWF PORTB
NOP	MOVWF PORTB	MOVWF PORTB	NOP
MOVLW B'10100111'	NOP	NOP	NOP
MOVWF PORTB	MOVLW B'00000100'	MOVLW B'11010110'	MOVWF PORTB
NOP	MOVWF PORTB	MOVWF PORTB	NOP
NOP	NOP	NOP	NOP
MOVLW B'11100101'	NOP	MOVLW B'01010110'	MOVWF PORTB
MOVWF PORTB	NOP	MOVWF PORTB	NOP
NOP	MOVLW B'00000100'	NOP	NOP
NOP	MOVWF PORTB	MOVLW B'10101110'	MOVWF PORTB
MOVLW B'11100101'	NOP	MOVWF PORTB	NOP
MOVWF PORTB	NOP	NOP	NOP
NOP	MOVLW B'10000100'	MOVLW B'01010110'	MOVWF PORTB
NOP	MOVWF PORTB	MOVWF PORTB	NOP
MOVLW B'01110101'	NOP	NOP	NOP
MOVWF PORTB	NOP	MOVLW B'01010110'	MOVWF PORTB
NOP	MOVLW B'10100100'	MOVWF PORTB	NOP
NOP	MOVWF PORTB	NOP	NOP
MOVLW B'01110101'	NOP	MOVLW B'00000011'	MOVWF PORTB
MOVWF PORTB	NOP	MOVWF PORTB	NOP
NOP	MOVLW B'11100010'	NOP	NOP
NOP	MOVWF PORTB	MOVLW B'00000011'	MOVWF PORTB
MOVLW B'00110000'	NOP	MOVWF PORTB	NOP
MOVWF PORTB	MOVLW B'11100010'	NOP	NOP
NOP	MOVWF PORTB	MOVLW B'10000011'	MOVWF PORTB
NOP	NOP	MOVWF PORTB	NOP
MOVLW B'00010000'	NOP	NOP	NOP
MOVWF PORTB	MOVLW B'01110010'	MOVLW B'10100011'	MOVWF PORTB
NOP	MOVWF PORTB	MOVWF PORTB	NOP
NOP	NOP	NOP	NOP
MOVLW B'10010000'	MOVLW B'01110010'	MOVLW B'10100011'	MOVWF PORTB
MOVWF PORTB	MOVWF PORTB	MOVWF PORTB	NOP
NOP	NOP	NOP	NOP
NOP	MOVLW B'01110010'	MOVLW B'11100001'	MOVWF PORTB
MOVLW B'10010000'	MOVWF PORTB	MOVWF PORTB	NOP
MOVWF PORTB	NOP	NOP	NOP
NOP	NOP	MOVLW B'11100001'	MOVWF PORTB
NOP	MOVLW B'00110011'	MOVWF PORTB	NOP
MOVLW B'11000010'	MOVWF PORTB	NOP	NOP
MOVWF PORTB	NOP	MOVLW B'11101001'	MOVWF PORTB
NOP	NOP	MOVWF PORTB	NOP
NOP	MOVLW B'00011011'	NOP	NOP
MOVLW B'11001010'	MOVWF PORTB	MOVLW B'01111001'	MOVWF PORTB
MOVWF PORTB	NOP	MOVWF PORTB	NOP
NOP	NOP	NOP	NOP
NOP	MOVLW B'10011011'	MOVLW B'01111001'	MOVWF PORTB
MOVLW B'01001010'	MOVWF PORTB	MOVWF PORTB	NOP
MOVWF PORTB	NOP	NOP	NOP
NOP	NOP	MOVLW B'00111000'	MOVWF PORTB
MOVLW B'01101010'	MOVWF PORTB	MOVWF PORTB	NOP
MOVWF PORTB	NOP	NOP	NOP
NOP	MOVLW B'11001001'	MOVLW B'00011000'	MOVWF PORTB
NOP	MOVWF PORTB	MOVWF PORTB	NOP
MOVLW B'00101011'	NOP	NOP	NOP
MOVWF PORTB	NOP	MOVLW B'11101010'	MOVWF PORTB
NOP	MOVLW B'11001001'	MOVWF PORTB	NOP
NOP	MOVWF PORTB	NOP	NOP
MOVLW B'00101011'	NOP	MOVLW B'00011000'	MOVWF PORTB
MOVWF PORTB	MOVLW B'11001001'	MOVWF PORTB	NOP
NOP	MOVWF PORTB	NOP	NOP
NOP	NOP	MOVLW B'01111110'	MOVWF PORTB
MOVLW B'10111111'	MOVWF PORTB	MOVWF PORTB	NOP
MOVWF PORTB	NOP	NOP	NOP
NOP	NOP	MOVLW B'01111110'	MOVWF PORTB
NOP	MOVLW B'01101101'	MOVWF PORTB	NOP

NOP	MOVLW B'11101101'	MOVWF PORTB	NOP
MOVLW B'00111111'	MOVWF PORTB	NOP	NOP
MOVWF PORTB	NOP	MOVLW B'00001100'	MOVWF PORTB
NOP	MOVLW B'11101101'	MOVWF PORTB	NOP
MOVLW B'00011111'	MOVWF PORTB	NOP	NOP
MOVWF PORTB	NOP	MOVLW B'01011110'	MOVWF PORTB
NOP	MOVLW B'01111101'	MOVWF PORTB	NOP
NOP	MOVWF PORTB	NOP	NOP
MOVLW B'10011111'	NOP	MOVLW B'10100100'	MOVWF PORTB
MOVWF PORTB	NOP	MOVWF PORTB	NOP
NOP	MOVLW B'01110101'	NOP	NOP
NOP	MOVWF PORTB	MOVLW B'11100110'	MOVWF PORTB
MOVLW B'10010111'	NOP	MOVWF PORTB	NOP
MOVWF PORTB	MOVLW B'00110100'	NOP	MOVLW B'00000111'
NOP	MOVWF PORTB	NOP	MOVWF PORTB
NOP	NOP	MOVLW B'11100110'	NOP
MOVLW B'11000101'	NOP	MOVWF PORTB	NOP
MOVWF PORTB	MOVLW B'00010100'	NOP	MOVLW B'00000111'
NOP	MOVWF PORTB	NOP	MOVWF PORTB
NOP	NOP	MOVLW B'11100110'	NOP
MOVLW B'11000101'	NOP	MOVWF PORTB	NOP
MOVWF PORTB	NOP	MOVLW B'01110110'	MOVWF PORTB
NOP	MOVLW B'10010100'	MOVWF PORTB	NOP
NOP	MOVWF PORTB	NOP	NOP
MOVLW B'01000101'	NOP	MOVLW B'01110110'	MOVWF PORTB
MOVWF PORTB	MOVLW B'10010100'	MOVWF PORTB	NOP
NOP	MOVWF PORTB	NOP	NOP
NOP	NOP	MOVLW B'00110011'	MOVWF PORTB
MOVLW B'01100101'	MOVWF PORTB	MOVWF PORTB	NOP
MOVWF PORTB	NOP	NOP	NOP
NOP	MOVLW B'11000010'	MOVLW B'00010011'	MOVWF PORTB
NOP	MOVWF PORTB	MOVWF PORTB	NOP
MOVLW B'00100000'	NOP	NOP	MOVLW B'11100001'
MOVWF PORTB	MOVLW B'11000010'	MOVWF PORTB	MOVWF PORTB
NOP	MOVWF PORTB	NOP	NOP
NOP	NOP	MOVLW B'10010011'	NOP
MOVLW B'00100000'	MOVWF PORTB	MOVWF PORTB	MOVLW B'01110001'
MOVWF PORTB	NOP	NOP	MOVWF PORTB
NOP	MOVLW B'01000010'	MOVLW B'10010011'	NOP
NOP	MOVWF PORTB	MOVWF PORTB	NOP
MOVLW B'10110000'	NOP	MOVLW B'10010011'	MOVWF PORTB
MOVWF PORTB	MOVWF PORTB	MOVWF PORTB	NOP
NOP	NOP	NOP	NOP
NOP	MOVLW B'01100010'	MOVLW B'11000001'	MOVWF PORTB
MOVLW B'10110000'	MOVWF PORTB	MOVWF PORTB	NOP
MOVWF PORTB	NOP	NOP	MOVLW B'00110000'
NOP	NOP	MOVLW B'11000001'	MOVWF PORTB
NOP	MOVLW B'00100011'	MOVWF PORTB	NOP
MOVLW B'11110010'	MOVWF PORTB	NOP	MOVLW B'00011000'
MOVWF PORTB	NOP	MOVLW B'11001001'	MOVWF PORTB
NOP	MOVLW B'00101011'	MOVWF PORTB	NOP
NOP	MOVWF PORTB	NOP	NOP
MOVLW B'11011010'	NOP	MOVLW B'01001001'	MOVWF PORTB
MOVWF PORTB	NOP	MOVWF PORTB	NOP
NOP	MOVLW B'10111011'	NOP	NOP
NOP	MOVWF PORTB	MOVLW B'01101001'	MOVWF PORTB
MOVLW B'01011010'	NOP	MOVWF PORTB	NOP
MOVWF PORTB	MOVLW B'10111011'	NOP	MOVLW B'11001010'
NOP	MOVWF PORTB	MOVLW B'00101000'	MOVWF PORTB
NOP	NOP	MOVWF PORTB	NOP
MOVLW B'01011010'	MOVLW B'11111001'	NOP	NOP
MOVWF PORTB	MOVWF PORTB	MOVLW B'11001010'	MOVWF PORTB
NOP	NOP	NOP	NOP
NOP	MOVLW B'11011001'	MOVLW B'00101000'	MOVWF PORTB
MOVLW B'00001011'	MOVWF PORTB	MOVWF PORTB	NOP
MOVWF PORTB	NOP	MOVLW B'11001010'	MOVWF PORTB
NOP	MOVLW B'11011001'	NOP	NOP
NOP	MOVWF PORTB	MOVLW B'01001110'	MOVWF PORTB
MOVLW B'00001011'	NOP	NOP	NOP
MOVWF PORTB	MOVLW B'01011101'	MOVLW B'01101110'	MOVWF PORTB
NOP	MOVWF PORTB	NOP	NOP
NOP	NOP	MOVLW B'10111100'	MOVWF PORTB
MOVLW B'10001111'	MOVWF PORTB	MOVWF PORTB	NOP
MOVWF PORTB	NOP	MOVLW B'10111100'	MOVWF PORTB
NOP	MOVLW B'01011101'	MOVWF PORTB	NOP
NOP	MOVWF PORTB	NOP	MOVLW B'00101111'
MOVLW B'10101111'	NOP	MOVLW B'11111110'	MOVWF PORTB
MOVWF PORTB	NOP	MOVWF PORTB	NOP
NOP	MOVLW B'00001100'		

```
NOP
MOVLW B'00101111'
MOVWF PORTB
NOP
NOP
MOVLW B'10111111'
MOVWF PORTB
NOP
```

```
NOP
MOVLW B'10110111'
MOVWF PORTB
NOP
NOP
MOVLW B'11110101'
MOVWF PORTB
NOP
```

```
NOP
MOVLW B'11010101'
MOVWF PORTB
NOP
NOP
MOVLW B'01010101'
MOVWF PORTB
NOP
```

```
NOP
MOVLW B'01010101'
MOVWF PORTB
GOTO LOOP
END
```

APPENDIX F

MICROCONTROLLER PROGRAM CODE FOR 16-ELECTRODE FREQUENCY DIVISION MULTIPLEXED IMPEDANCE TOMOGRAPHY SYSTEM

```
        INCLUDE <PIC1684.SRC>
        BSF STATUS, RP0
        CLRF TRISB
        BCF STATUS, RP0
        CLRF PORTB
LOOP    MOVLW B'00000010'
        MOVWF PORTB
        NOP
        NOP
        MOVLW B'00000011'
        MOVWF PORTB
        NOP
        NOP
        MOVLW B'00000001'
        MOVWF PORTB
        NOP
        NOP
        MOVLW B'00000000'
        MOVWF PORTB
        GOTO LOOP
        END
```

APPENDIX G FUNDAMENTAL FREQUENCIES AND HARMONICS FOR 8-ELECTRODE FREQUENCY DIVISION MULTIPLEXED IMPEDANCE TOMOGRAPHY SYSTEM

HARMONIC	TxA	TxB	TxC	TxD
FUNDAMENTAL	25kHz	31.25kHz	41.67kHz	62.5kHz
3rd	75kHz	93.75kHz	125kHz	187.5kHz
5th	125kHz	156.25kHz	208.33kHz	312.5kHz
7th	175kHz	218.75kHz	291.67kHz	437.5kHz
9th	225kHz	281.25kHz	375kHz	562.5kHz

APPENDIX H

FUNDAMENTAL FREQUENCIES AND HARMONICS FOR 16-ELECTRODE FREQUENCY DIVISION MULTIPLEXED IMPEDANCE TOMOGRAPHY SYSTEM

HARMONIC	TxA	TxB	TxC	TxD	TxE	TxF	TxG	TxH
FUNDAMENTAL	79.17kHz	75kHz	73.53kHz	66.67kHz	62.84kHz	54.35kHz	23.44kHz	48kHz
3rd	237.52kHz	225kHz	220.59kHz	200kHz	188.51kHz	163.04kHz	70.31kHz	144kHz
5th	395.86kHz	375kHz	367.65kHz	333.33kHz	314.18kHz	271.17kHz	117.19kHz	240kHz
7th	554.20kHz	525kHz	514.71kHz	466.67kHz	439.85kHz	380.43kHz	164.06kHz	336kHz
9th	712.55kHz	675kHz	661.76kHz	600kHz	565.53kHz	489.13kHz	210.94kHz	432kHz

APPENDIX I

AUTOMATIC CODE GENERATOR PROGRAM CODE

PROGRAM LISTING OF AUTONEW.PAS

(* The program automatically generates the program code for the PIC16F84 to achieve a specific division factor. The code is saved to a temporary textfile, compiled using MPASM and downloaded using the software provided with the PIC programmer. The output square waves are generated using the following sequence of bit sets

```
PORTB0 0   1   1   0
PORTB1 1   1   0   0      *)
```

unit autonew;

interface

uses

Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs, StdCtrls, Spin;

type

```
TAutoMain = class(TForm)
  Label1: TLabel;
  DivSelect: TSpinEdit;
  Generate: TButton;
  Compile: TButton;
  Download: TButton;
  SaveDialog: TSaveDialog;
  procedure GenerateClick(Sender: TObject); // generate code sequence
  procedure CompileClick(Sender: TObject); // compile code sequence
  procedure DownloadClick(Sender: TObject); // download code sequence
private
  { Private declarations }
public
  { Public declarations }
end;
```

var

AutoMain: TAutoMain;

implementation

{ \$R *.DFM }

// generate sequence of instructions to achieve a specific division factor -
 // the number of included 'NOP' instructions sets this factor. The first three
 // instructions also require two 'NOP's to compensate for the instruction
 // cycles to loop back to the start after the fourth instruction - this
 // ensures that the duty cycle is 50% and 90 degrees of phase shift is
 // achieved.

procedure TAutoMain.GenerateClick(Sender: TObject);

var tempfile : textfile;

numextra, i : integer;

begin

if SaveDialog.Execute then

begin

// calculate the number of extra 'NOP's required to achieve

// a division factor

numextra := ((DivSelect.Value div 16) - 4);

AssignFile(tempfile, SaveDialog.FileName);

ReWrite(tempfile);

// configure PORTB as an output

WriteLn(tempfile, #9 + 'INCLUDE <PIC1684.SRC>');

WriteLn(tempfile, #9 + 'BSF STATUS, RP0');

WriteLn(tempfile, #9 + 'CLRF TRISB');

WriteLn(tempfile, #9 + 'BCF STATUS, RP0');

WriteLn(tempfile, #9 + 'CLRF PORTB');

WriteLn(tempfile, 'LOOP' + #9 + 'MOVLW B'00000010'');

WriteLn(tempfile, #9 + 'MOVWF PORTB');

WriteLn(tempfile, #9 + 'NOP');

WriteLn(tempfile, #9 + 'NOP');

for i := 1 to numextra do

WriteLn(tempfile, #9 + 'NOP');

WriteLn(tempfile, #9 + 'MOVLW B'00000011'');

WriteLn(tempfile, #9 + 'MOVWF PORTB');

WriteLn(tempfile, #9 + 'NOP');

WriteLn(tempfile, #9 + 'NOP');

for i := 1 to numextra do

WriteLn(tempfile, #9 + 'NOP');

WriteLn(tempfile, #9 + 'MOVLW B'00000001'');

WriteLn(tempfile, #9 + 'MOVWF PORTB');

WriteLn(tempfile, #9 + 'NOP');

WriteLn(tempfile, #9 + 'NOP');

for i := 1 to numextra do

WriteLn(tempfile, #9 + 'NOP');

WriteLn(tempfile, #9 + 'MOVLW B'00000000'');

WriteLn(tempfile, #9 + 'MOVWF PORTB');

for i := 1 to numextra do

WriteLn(tempfile, #9 + 'NOP');

WriteLn(tempfile, #9 + 'GOTO LOOP');

WriteLn(tempfile, #9 + 'END');

CloseFile(tempfile);

end;

end;

// compile the text file

procedure TAutoMain.CompileClick(Sender: TObject);

var tempstring : string;

begin

tempstring := 'MPASM ' + SaveDialog.FileName + ' /pPIC16F84 /aINHX8M';

WinExec(PChar(tempstring), SW_SHOWMAXIMIZED);

end;

// download the text file

procedure TAutoMain.DownloadClick(Sender: TObject);

begin

WinExec('PP', SW_SHOWMAXIMIZED);

end;

end.

APPENDIX J

PBIL OPTIMISATION ALGORITHM AND SIMULATOR PROGRAM CODE

PROGRAM LISTING OF PBILFORM.PAS

(* The following application performs PBIL to determine the set of optimal frequencies for an 8-electrode FDM impedance tomography system. The program uses all the standard crystal frequencies and all division factors that can be programmed in the PIC16F84A to determine a set of possible frequencies between 17.6kHz and 80kHz. A trial solution is taken from this set and the process of mixing in the synchronous detectors is simulated as a sum and difference of the frequency harmonics. The aliasing that occurs in the switched capacitor filters on the output is also simulated and the fitness of a trial solution is the minimum frequency component at the LPF outputs. Clearly the objective is to maximise this frequency so that all frequency components are outside the passband of the LPF and can therefore be filtered out. *)

unit pbilform;

interface

uses

Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs, StdCtrls, Math, ComCtrls, Spin, ExtCtrls;

type

TPBILMain = class(TForm)
 LoadFreqs: TButton;
 TestMult: TButton;
 Label1: TLabel;
 Label2: TLabel;
 MaxGenerations: TSpinEdit;
 NumTrials: TSpinEdit;
 PBILStart: TButton;
 PBILProgress: TProgressBar;
 Freq1: TLabel;
 Freq2: TLabel;
 Freq3: TLabel;
 Freq4: TLabel;
 Freq5: TLabel;
 Freq6: TLabel;
 Freq7: TLabel;
 Freq8: TLabel;
 Label3: TLabel;
 BestFreqFit: TLabel;
 MultiSearchPBIL: TButton;
 Label4: TLabel;
 NumIterations: TSpinEdit;
 Label5: TLabel;
 CurrIterCaption: TLabel;
 Label6: TLabel;
 OverBestFreqFit: TLabel;
 OverFreq5: TLabel;
 OverFreq6: TLabel;
 OverFreq7: TLabel;
 OverFreq8: TLabel;
 OverFreq1: TLabel;
 OverFreq2: TLabel;
 OverFreq3: TLabel;
 OverFreq4: TLabel;
 SaveMultiRecord: TSaveDialog;
 Label7: TLabel;
 Label8: TLabel;
 FiltCap2: TEdit;
 FiltRes2: TEdit;
 Label9: TLabel;
 Label10: TLabel;
 FiltRes1: TEdit;
 FiltCap1: TEdit;
 MagThresh: TEdit;
 Label11: TLabel;
 Bevel1: TBevel;
 procedure LoadFreqsClick(Sender: TObject);
 procedure FormShow(Sender: TObject);

 procedure TestMultClick(Sender: TObject);
 procedure PBILStartClick(Sender: TObject);
 procedure MultiSearchPBILClick(Sender: TObject);
private
 { Private declarations }
 xtal : array[1..23] of double; // standard crystal frequencies
 AvailFreq : array[1..2000] of double; // available frequencies
 RecvFreq : array[1..2000] of double; // received frequencies
 MultOutFreqs : array[1..200000] of double; // multiplier output frequencies
 resrecord : array[1..9,1..100] of double; // record of results
 FreqMag : array[1..801,1..2] of extended; // record of simulated frequency
 // harmonics and corresponding
 // magnitudes

 ntrials : integer;
 maxgen : integer;
 besteverfit : double;
 res1 : double; // component parameters of the receiver filters
 cap1 : double;
 res2 : double;
 cap2 : double;
 TC1 : extended;
 TC2 : extended;
 gain1 : extended;
 gain2 : extended;
 temp1, temp2, temp3, temp4, temp5 : extended;
 diffrec : array [0..7] of double;
 RefFreq, clock : double;
 RefHarm : array [1..51] of double;
 threshold : double; // magnitude threshold - all recieved frequency
 // components whose magnitudes are less than
 // the threshold are ignored
 besteverreal : array[1..8] of double;
 function MultOut(inputs:array of double):double;
public
 { Public declarations }
end;

var

 PBILMain: TPBILMain;

implementation

 {\$R *.DFM}

 // Quick sort routine to sort an array in ascending order
 // (adapted from Borland's BP7 demo)
 procedure QSort(var X : array of double; Lbound, Ubound : Integer);
 procedure Sort(L, R : Integer);
 var
 I, J : integer;
 U, V : double;
 begin
 I := L;
 J := R;
 U := X[(L + R) div 2];
 repeat
 while X[I] < U do I := I + 1;
 while U < X[J] do J := J - 1;
 if I <= J then
 begin
 V := X[I]; X[I] := X[J]; X[J] := V;
 I := I + 1; J := J - 1;
 end;
 until I > J;
 if L < J then Sort(L, J);
 if I < R then Sort(I, R);
 end;
 begin
 Sort(Lbound, Ubound);
 end;

```

// ----- frequency simulation procedures ----- //
// generate a list of all available transmitter frequencies
procedure TPBILMain.LoadFreqsClick(Sender: TObject);
var i,j,k : integer;
    factor : real;
    divfactor : array[1..60] of double;
    TempFreq : array[1..2000] of double;
begin
    factor := 64;
    i := 1;
    while (factor <= 1008) do
    begin
        divfactor[i] := 1/factor;
        inc(i);
        factor := factor + 16;
    end;

    i := 1;
    for j:= 1 to 23 do
    begin
        for k:= 1 to 60 do
        begin
            TempFreq[i] := xtal[j]*divfactor[k];
            inc(i);
        end;
    end;

    i := 1;
    j := 1;
    while (TempFreq[i] <> 0) do
    begin
        if ((TempFreq[i] >= 17.6e3) and (TempFreq[i] <= 80e3)) then
        begin
            AvailFreq[j] := TempFreq[i];
            inc(j);
        end;
        inc(i);
    end;

    QSort(AvailFreq,0,j-2);
    k := 1;
    for i := 1 to j-1 do
    begin
        if (AvailFreq[i] <> AvailFreq[i+1]) then
        begin
            TempFreq[k] := AvailFreq[i];
            inc(k);
        end;
    end;
    for i := 1 to k-1 do
    begin
        AvailFreq[i] := TempFreq[i];
    end;
end;

// setup array of all standard crystal frequencies
procedure TPBILMain.FormShow(Sender: TObject);
begin
    xtal[1] := 1e6;
    xtal[2] := 1.6384e6;
    xtal[3] := 1.8432e6;
    xtal[4] := 2e6;
    xtal[5] := 2.4576e6;
    xtal[6] := 3e6;
    xtal[7] := 3.2768e6;
    xtal[8] := 3.579545e6;
    xtal[9] := 3.6864e6;
    xtal[10] := 4e6;
    xtal[11] := 4.433619e6;
    xtal[12] := 5e6;
    xtal[13] := 5.0688e6;
    xtal[14] := 6e6;
    xtal[15] := 6.144e6;
    xtal[16] := 8e6;
    xtal[17] := 8.867238e6;
    xtal[18] := 10e6;
    xtal[19] := 11.0592e6;
    xtal[20] := 12e6;
    xtal[21] := 16e6;
    xtal[22] := 16.38e6;

```

```

    xtal[23] := 20e6;
end;

// calculate the minimum frequency component on the LPF outputs by simulating
// the mixing in the synchronous detectors and aliasing in the switched capacitor
// filters for a set of transmitter frequencies. The magnitudes are the frequency
// harmonics are simulated so only those frequencies with a magnitude greater than
// the threshold are included.
function TPBILMain.MultOut(inputs:array of double);double;
var a,b,c,d,e, upper, numfact: integer;
begin
    Qsort(inputs,0,7);

    // include a DC component with a magnitude of 0.5
    FreqMag[1,1] := 0;
    FreqMag[1,2] := 0.5;
    b := 2;

    // calculate up to the 100th harmonic of the transmitter but only
    // include those frequency harmonics whose magnitudes after the
    // filtering in the receiver are greater than the minimum
    for c:= 1 to 100 do
        for a:= 0 to 7 do
        begin
            // calculate the frequency by multiplying a transmitter
            // frequency by an integer factor
            FreqMag[b,1] := inputs[a]*c;
            temp1 := inputs[a]*c;
            temp2 := TC1*temp1;
            temp3 := TC2*temp1;
            temp4 := sqr(temp2);
            temp5 := sqr(temp3);
            // calculate the attenuation due to filter 1
            gain1 := 1/sqrt(1+temp4);
            // calculate the attenuation due to filter 2
            gain2 := 1/sqrt(1+temp5);
            // the magnitude of a square wave harmonic is given by
            // 1/c*pi where c is the order of the harmonic
            // the magnitude of the received harmonic is then
            // this magnitude multiplied by the gains of the two
            // filter stages
            FreqMag[b,2] := gain1*gain2*(1/(c*pi));

            inc(b);
        end;
    end;

    e := 1;
    for c := 1 to 800 do
    begin
        // only frequencies larger than the threshold are kept
        if FreqMag[c,2] > threshold then
        begin
            RecvFreq[e] := FreqMag[c,1];
            inc(e);
        end;
    end;

    // sort the received frequencies and check for duplicates - if there are
    // any duplicate frequencies then one transmitter is some multiple of
    // another and therefore the minimum frequency component is 0Hz
    upper := e;
    QSort(RecvFreq,0,e-2);
    a := 2;
    while(RecvFreq[a] <> 0) do
    begin
        if (RecvFreq[a] = RecvFreq[a+1]) then
        begin
            MultOut := 0;
            exit;
        end;
        inc(a);
    end;

    // for all 8 transmitter frequencies
    for a := 0 to 7 do
    begin
        // calculate the harmonics of the reference square wave - since
        // it is a perfect square wave (TTL thresholding), include only
        // odd harmonics
        RefFreq := inputs[a];
        RefHarm[1] := 0;

```



```

c := 1;
for b := 2 to 51 do
begin
    RefHarm[b] := RefFreq*c;
    inc(c,2);
end;

// simulate the multiplier output frequencies as a sum and
// difference of the received and reference frequencies
d := 1;
for c := 1 to 51 do
    for b := 1 to (upper - 1) do
        begin
            MultOutFreqs[d] := RefHarm[c]+RecvFreq[b];
            inc(d);
            MultOutFreqs[d] := abs(RefHarm[c]-RecvFreq[b]);
            inc(d);
        end;
    end;

Qsort(MultOutFreqs,0,d-2);

// clock frequency is generally the lowest transmitter frequency
clock := inputs[0];

// simulate the aliasing in the switched capacitor filters
for c := 1 to d-1 do
begin
    if MultOutFreqs[c] >= clock then
        begin
            numfact := floor(MultOutFreqs[c]/clock);
            MultOutFreqs[c] := MultOutFreqs[c]-numfact*clock;
        end;
    if MultOutFreqs[c] >= clock/2 then
        MultOutFreqs[c] := clock - MultOutFreqs[c];
    end;

c := 1;
while (MultOutFreqs[c] < 0.1) do
    inc(c);

    diffrec[a] := MultOutFreqs[c];
end;
Qsort(diffrec,0,7);
MultOut := diffrec[0];
end;

// check the performance of a set of transmitter frequencies
procedure TPBILMain.TestMultClick(Sender: TObject);
var input : array[1..8] of double;
    res : double;
begin
    threshold := strtfloat(MagThresh.Text);
    res1 := strtfloat(FiltRes1.Text);
    cap1 := strtfloat(FiltCap1.Text);
    res2 := strtfloat(FiltRes2.Text);
    cap2 := strtfloat(FiltCap2.Text);
    TC1 := res1*cap1*2*pi;
    TC2 := res2*cap2*2*pi;
    input[1] := 23437.275;
    input[2] := 48005.1375;
    input[3] := 54346.8875;
    input[4] := 62829.1;
    input[5] := 66692.8125;
    input[6] := 73528.525;
    input[7] := 74997.1875;
    input[8] := 79172.8125;
    res := MultOut(input);
    BestFreqFit.Caption := floattostrf(res,ffFixed,5,4);
end;

// ----- PBIL procedures ----- //
// perform PBIL to determine optimal set of transmitter frequencies for
// an 8-electrode FDM impedance tomography system
procedure TPBILMain.PBILStartClick(Sender: TObject);
var nvars, prec, g, trial, i,j,k : integer;
    bestfit, fit : double;
    bw : array[1..8] of integer;
    PV : array[1..64] of double;
    R : array[1..8] of integer;
    bestreal : array[1..8] of double;
    trialsol : array[1..64] of short;

bestsol : array[1..64] of short;
besteversol : array[1..64] of short;
inputs : array[1..8] of double;
res : double;
begin
    res1 := strtfloat(FiltRes1.Text);
    cap1 := strtfloat(FiltCap1.Text);
    res2 := strtfloat(FiltRes2.Text);
    cap2 := strtfloat(FiltCap2.Text);
    TC1 := res1*cap1*2*pi;
    TC2 := res2*cap2*2*pi;
    threshold := strtfloat(MagThresh.Text);
    // initialise the random number generator with a random value
    Randomize;
    ntrials := NumTrials.Value;
    maxgen := MaxGenerations.Value;
    PBILProgress.Position := 0;
    PBILProgress.Max := maxgen + 1;
    nvars := 8; // number of variables
    prec := 8; // 8-bit precision since want to index into
                // an array 256 elements long to select frequency
    besteverfit := -1; // record of best ever result

    // calculate the bit weights
    for g := 0 to prec-1 do
        bw[g+1] := floor(power(2,g));
    // initialise the probability vector to all 0.5
    for g := 1 to prec*nvars do
        PV[g] := 0.5;

    for g := 1 to maxgen do
        begin
            PBILProgress.StepIt;
            bestfit := -1;
            for trial := 1 to ntrials do
                begin
                    // generate a random trial solution
                    for i:= 1 to (prec*nvars) do
                        trialsol[i] := short(PV[i] > Random);

                    // convert the sequence of bits into integer values
                    k := 1;
                    for i:= 1 to nvars do
                        begin
                            R[i] := 0;
                            for j := 1 to prec do
                                begin
                                    R[i] := R[i]+trialsol[k]*bw[j];
                                    inc(k);
                                end;
                            end;

                        end;

                    // use these 8 integer values to index into the array
                    // of available transmitter frequencies
                    for i := 1 to nvars do
                        inputs[i] := AvailFreq[R[i]+1];

                    // test current set of 8 transmitter frequencies
                    fit := MultOut(inputs);

                    // if the performance is the best so far for this
                    // generation then update the record
                    if fit > bestfit then
                        begin
                            bestfit := fit;
                            for i:= 1 to (nvars*prec)do
                                begin
                                    bestsol[i] := trialsol[i];
                                    if (i<=nvars) then
                                        bestreal[i] := inputs[i];
                                    end;
                                end;
                            end;
                        end;

                    res := MultOut(bestreal);
                    if res > 0.1 then
                        begin
                            // using the best solution, adjust the probability vector
                            // towards the best solution
                            for i:=1 to (prec*nvars) do

```

```

begin
    PV[i] := 0.9*PV[i]+0.1*bestsol[i];
    PV[i] := PV[i]-0.005*(PV[i]-0.5);
end;

// keep record of the best overall results
if bestfit > besteverfit then
begin
    besteverfit := bestfit;
    res := MultOut(bestreal);
    BestFreqFit.Caption := floattostrf(res,ffFixed,15,5);
    for i:= 1 to (nvars*prec)do
        begin
            besteversol[i] := bestsol[i];
            if (i<=nvars) then
                besteverreal[i] := bestreal[i];
            end;
        end;
    Qsort(besteverreal,0,7);
    Freq1.Caption := floattostrf(besteverreal[1],ffFixed,15,4);
    Freq2.Caption := floattostrf(besteverreal[2],ffFixed,15,4);
    Freq3.Caption := floattostrf(besteverreal[3],ffFixed,15,4);
    Freq4.Caption := floattostrf(besteverreal[4],ffFixed,15,4);
    Freq5.Caption := floattostrf(besteverreal[5],ffFixed,15,4);
    Freq6.Caption := floattostrf(besteverreal[6],ffFixed,15,4);
    Freq7.Caption := floattostrf(besteverreal[7],ffFixed,15,4);
    Freq8.Caption := floattostrf(besteverreal[8],ffFixed,15,4);
end;
end;
Application.ProcessMessages;
end;
end;

// since PBIL is inherently random in nature, the first result achieved is not
// necessarily the best therefore the process should be repeated multiple times
// and the best results then used - this procedure simply implements multiple
// PBIL searches keeping track of the best results
procedure TPBILMain.MultiSearchPBILClick(Sender: TObject);
var currentiter, i : integer;
    overbestfreq : double;
    multirecord : textfile;
    filestring : string;
begin
    LoadFreqsClick(Sender);
    overbestfreq := -1;
    for currentiter := 1 to NumIterations.Value do
        begin
            CurlterCaption.Caption := inttostr(currentiter);
            PBILStartClick(Sender);

```

```

            if besteverfit > overbestfreq then
                begin
                    overbestfreq := besteverfit;
                    OverFreq1.Caption := Freq1.Caption;
                    OverFreq2.Caption := Freq2.Caption;
                    OverFreq3.Caption := Freq3.Caption;
                    OverFreq4.Caption := Freq4.Caption;
                    OverFreq5.Caption := Freq5.Caption;
                    OverFreq6.Caption := Freq6.Caption;
                    OverFreq7.Caption := Freq7.Caption;
                    OverFreq8.Caption := Freq8.Caption;
                    OverBestFreqFit.Caption := BestFreqFit.Caption;
                end;

            resrecord[1,currentiter] := besteverfit;
            for i := 2 to 9 do
                resrecord[i,currentiter] := besteverreal[i-1];

            filestring := 'c:\multi'+inttostr(currentiter)+'.txt';
            AssignFile(multirecord,filestring);
            ReWrite(multirecord);
            WriteLn(multirecord,'Best fit: '+floattostr(resrecord[1,currentiter]));
            WriteLn(multirecord,'Frequencies for best fit:');
            for i := 2 to 9 do
                WriteLn(multirecord,floattostr(resrecord[i,currentiter]));
            WriteLn(multirecord,' ');
            end;
            CloseFile(multirecord);
        end;
    end;
end;

if SaveMultiRecord.Execute then
begin
    AssignFile(multirecord,SaveMultiRecord.FileName);
    ReWrite(multirecord);
    for currentiter := 1 to NumIterations.Value do
        begin
            WriteLn(multirecord,'Best fit: '+floattostr(resrecord[1,currentiter]));
            WriteLn(multirecord,'Frequencies for best fit:');
            for i := 2 to 9 do
                WriteLn(multirecord,floattostr(resrecord[i,currentiter]));
            WriteLn(multirecord,' ');
            end;
            CloseFile(multirecord);
        end;
    end;
end.

```

APPENDIX K

SAMPLER PROGRAM CODE

PROGRAM LISTING OF MAINUNIT.PAS

(* This program is used to capture training data from the impedance tomography system in a static situation. The desired network output for each test case is specified through an InputTomo component, which allows the user to set the phase of each pixel in an image representation of the vessel cross-section. The voltages are captured from the system using the sampunit and the results are stored in a Delphi database. A database consists of two tables, namely InpTable (for the 128 voltage readings from the rig) and OutTable (for the pixels specifying the desired network output and the volume fractions for the phases). An option is provided to continuously sample data from the system and display the results in a tabular format - this will be useful for de-bugging the final system since the user can monitor all 128 voltages without having to measure at that particular point on the board using a multimeter. *)

unit mainunit;

interface

uses

Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs, ExtCtrls, Tomo, Menus, StdCtrls, DBTables, DB, Buttons, sampunit, Spin, Registry;

type

TTomoSystem = (top, bottom); // dedicted type to differentiate
// between readings from the top
// and bottom system

TSampMain = class(TForm)

SampMainMenu: TMainMenu;

File1: TMenuItem;

Opendatabase1: TMenuItem;

Createdatabase1: TMenuItem;

Closedatabase1: TMenuItem;

N1: TMenuItem;

Exit1: TMenuItem;

View1: TMenuItem;

Startsampling1: TMenuItem;

View2: TMenuItem;

Database1: TMenuItem;

Currentsamples1: TMenuItem;

Panel1: TPanel;

Session: TSession;

SampAliasOpen: TPanel;

Label1: TLabel;

SampAliasList: TComboBox;

SampAliasNew: TPanel;

Label2: TLabel;

SampAliasNewName: TEdit;

SampAliasNameOK: TBitBtn;

SampAliasNewCanc: TBitBtn;

SampAliasOpenCanc: TBitBtn;

Setthenumberofframes1: TMenuItem;

SampNumFrames: TPanel;

Label3: TLabel;

SampNumFramesValue: TSpinEdit;

SampNumFramesOK: TBitBtn;

SampNumFramesCan: TBitBtn;

procedure Exit1Click(Sender: TObject);

procedure Opendatabase1Click(Sender: TObject);

procedure SampAliasListClick(Sender: TObject);

procedure Closedatabase1Click(Sender: TObject);

procedure Createdatabase1Click(Sender: TObject);

procedure SampAliasNameOKClick(Sender: TObject);

procedure SampAliasNewCancClick(Sender: TObject);

procedure SampAliasOpenCancClick(Sender: TObject);

procedure Startsampling1Click(Sender: TObject);

procedure FormCreate(Sender: TObject);

procedure FormClose(Sender: TObject; var Action: TCloseAction);

procedure Database1Click(Sender: TObject);

procedure Currentsamples1Click(Sender: TObject);

procedure SampNumFramesOKClick(Sender: TObject);

procedure SampNumFramesCanClick(Sender: TObject);

procedure Setthenumberofframes1Click(Sender: TObject);

private

{ Private declarations }

dbasename : string; // store the name of the current
// database so that it can be written
// to the registry on exit

procedure CreateTable(aliasname: string);

// initialise the tables (and create the
// tables if they do not already exist)

public

{ Public declarations }

end;

var

SampMain: TSampMain;

InpTable: TTable; // table to store input data

OutTable: TTable; // table to store desired output

SampHardware: TSamp; // object used to control sampling

notifvalue : cardinal; // Windows notification value for

// streaming complete

result : array[0..65535] of double; // temporary array to store streaming
// results

numframes : integer; // number of frames of data to capture

implementation

uses addunit, curunit;

{ \$R *.DFM }

(* ----- DATABASE PROCEDURES ----- *)

(* get a list of all the available databases - the user then selects one *)

procedure TSampMain.Opendatabase1Click(Sender: TObject);

var templist: TStringList;

i : integer;

begin

templist := TStringList.Create;

Session.GetAliasNames(templist);

for i := 0 to templist.Count - 1 do

SampAliasList.Items.Add(templist[i]);

SampAliasOpen.Visible := true;

templist.Free;

end;

(* when the user selects a specific database, initialise the tables *)

procedure TSampMain.SampAliasListClick(Sender: TObject);

begin

SampAliasOpen.Visible := false;

if SampAliasList.Text <> '' then

CreateTables(SampAliasList.Text);

end;

(* close the databases *)

procedure TSampMain.Closedatabase1Click(Sender: TObject);

begin

InpTable.Close;

OutTable.Close;

end;

(* allow the user to create a new database *)

procedure TSampMain.Createdatabase1Click(Sender: TObject);

begin

SampAliasNew.Visible := true;

SampAliasNewName.SetFocus;

end;

(* once the user has entered a name for their new database, create a directory

for the tables, create a database alias and initialise the tables *)

procedure TSampMain.SampAliasNameOKClick(Sender: TObject);

var directory: string;

begin

directory := 'c:\Courses\Masters\' + SampAliasNewName.Text;

if Session.IsAlias(SampAliasNewName.Text) then

// check to see whether the database already exists

MessageDlg('Database already exists', mtError, [mbOK], 0)

```

else
begin
    // else add the database
    CreateDirectory(PChar(directory),nil);
    Session.AddStandardAlias(SampAliasNewName.Text,directory,'PARADOX');
    Session.SaveConfigFile;
    CreateTables(SampAliasNewName.Text);
end;
SampAliasNew.Visible := false;
SampAliasNewName.Text := "";
end;

(* create the tables and set them for the current database. If the tables don't
exist, then specify the table fields, the table indeces and add an empty record
to initialise the table for use. If the tables exist, then simply open the tables
and allow editing of the tables *)
procedure TSampMain.CreateTables(aliasname : string);
var i : integer;
    s : string;
begin
    dbasename := aliasname;
    InpTable := TTable.Create(self);
    OutTable := TTable.Create(self);
    InpTable.DatabaseName := aliasname;
    OutTable.DatabaseName := aliasname;
    InpTable.TableName := 'InpTrain.db';
    OutTable.TableName := 'OutTrain.db';

    // if the tables do not exist
    if not(InpTable.Exists and OutTable.Exists) then
    begin
        // setup the fields for the tables
        InpTable.FieldDefs.Add('TestNo',ftFloat);
        InpTable.FieldDefs.Add('System',ftInteger);
        OutTable.FieldDefs.Add('TestNo',ftFloat);
        // fields for 128 input voltages obtained from the system
        for i := 1 to 128 do
        begin
            s := 'Voltage'+inttostr(i);
            InpTable.FieldDefs.Add(s,ftFloat);
        end;
        // fields for the desired volume fractions for specific
        // test cases
        OutTable.FieldDefs.Add('WaterVolume',ftFloat);
        OutTable.FieldDefs.Add('GravelVolume',ftFloat);
        OutTable.FieldDefs.Add('AirVolume',ftFloat);
        // fields for output pixel values for specific test cases
        for i := 1 to 100 do
        begin
            s := 'Pixel'+inttostr(i);
            OutTable.FieldDefs.Add(s,ftSmallInt)
        end;

        // for all tables, the testno will be used as the index
        InpTable.IndexDefs.Add('TestNo',[ixPrimary]);
        InpTable.CreateTable;
        OutTable.IndexDefs.Add('TestNo',[ixPrimary]);
        OutTable.CreateTable;

        // add an empty record to the table to initialise the table
        // for further use
        InpTable.Open;
        OutTable.Open;
        InpTable.Edit;
        OutTable.Edit;
        InpTable.Append;
        InpTable.FieldByName('TestNo').Value := 0;
        OutTable.Append;
        OutTable.FieldByName('TestNo').Value := 0;
        InpTable.Post;
        OutTable.Post;
    end
    else
    begin
        // else simply open the tables and allow editing
        InpTable.Open;
        OutTable.Open;
        InpTable.Edit;
        OutTable.Edit;
    end;
end;

end;

end;

(* ----- GENERAL PROCEDURES ----- *)
(* exit the program *)
procedure TSampMain.Exit1Click(Sender: TObject);
begin
    Close;
end;

(* if the user changes his mind *)
procedure TSampMain.SampAliasNewCancClick(Sender: TObject);
begin
    SampAliasNewName.Text := "";
    SampAliasNew.Visible := false;
end;

procedure TSampMain.SampAliasOpenCancClick(Sender: TObject);
begin
    SampAliasOpen.Visible := false;
end;

(* to start sampling, initialise the sample hardware and display AddToDatabase
modally *)
procedure TSampMain.Startsampling1Click(Sender: TObject);
begin
    // store the notification value for the sample unit so that
    // can capture Windows notification message indicating that
    // sampling is complete
    notifvalue := SampHardware.InitSample(1,AddToDatabase.Handle);
    AddToDatabase.Caption := 'Add sampled data to database';
    // allow editing of data
    AddToDatabase.AddInputTomo.AllowEdit := true;
    AddToDatabase.AddSampleStart.Visible := true;
    AddToDatabase.AddReset.Visible := true;
    AddToDatabase.AddUserSetVolume.Visible := true;
    AddToDatabase.ShowModal;
end;

(* when start the program, read from the registry to determine which was the
last database to be used and how many frames were taken for each test case.
Load the last database to be used automatically *)
procedure TSampMain.FormCreate(Sender: TObject);
var Reg : TRegistry;
    KeyGood : Boolean;
begin
    SampHardware := TSamp.Create;
    Reg := TRegistry.Create;
    try
        KeyGood := Reg.OpenKey('Software\Sampler', False);
        // if the key exists then read from the key, else it is the
        // first time the program is running so do not attempt
        // to read from the registry
        if KeyGood then
        begin
            numframes := Reg.ReadInteger('NumFrames');
            SampNumFramesValue.Value := numframes;
            dbasename := Reg.ReadString('Database');
            // automatically load the last database to be used
            CreateTables(dbasename);
        end;
    finally
        Reg.Free;
    end;
end;

(* when close program, release the resources allocated for sampling and write
to the registry the current configuration *)
procedure TSampMain.FormClose(Sender: TObject; var Action: TCloseAction);
var Reg: TRegistry;
begin
    SampHardware.ReleaseSample;
    SampHardware.Free;
    Reg := TRegistry.Create;
    try
        Reg.OpenKey('Software\Sampler', True);
        Reg.WriteInteger('NumFrames', numframes);
        Reg.WriteString('Database',dbasename);
    finally
        Reg.Free;
    end;
end;
end;

```

```
(* show AddToDatabase modally to view the data currently in the database *)
procedure TSampMain.Database1Click(Sender: TObject);
begin
    AddToDatabase.Caption := 'View database data';
    // stop editing of data - purely for viewing sampled data
    AddToDatabase.AddInputTomo.AllowEdit := false;
    AddToDatabase.AddSampleStart.Visible := false;
    AddToDatabase.AddReset.Visible := false;
    AddToDatabase.AddUserSetVolume.Visible := false;
    AddToDatabase.AddSetGravel.Enabled := false;
    AddToDatabase.AddSetAir.Enabled := false;
    AddToDatabase.ShowModal;
end;

(* to continuously sample the system, initialise the sampling hardware
and show CurMain modally *)
procedure TSampMain.Currentsamples1Click(Sender: TObject);
begin
    notifvalue := SampHardware.InitSample(1,CurMain.Handle);
    CurMain.ShowModal;
end;

(* set the number of frames *)
procedure TSampMain.SampNumFramesOKClick(Sender: TObject);
begin
    numframes := SampNumFrames.Value.Value;
    SampNumFrames.Visible := false;
end;

(* if the user changes his mind *)
procedure TSampMain.SampNumFramesCanClick(Sender: TObject);
begin
    SampNumFrames.Visible := false;
end;

procedure TSampMain.Setthenumberofframes1Click(Sender: TObject);
begin
    SampNumFrames.Visible := true;
end;

end.
```

PROGRAM LISTING OF ADDUNIT.PAS

(* This unit is used to capture samples from the impedance tomography system and store them in the database. Also stored in the database is a cross-sectional image of the pipe at the time of the test. This image is set using a Tomo component, which I created specifically for this task. Buttons are also provided which permit the user to look at previous test cases in the database. To create a new test case, move to the end of the database and click the 'RIGHT' arrow. This will automatically create space in the database for the new test case. The user then specifies what the desired output for that test case is and presses the 'Start sampling.' button to initiate the sampling as a background process. The results of the current sample are displayed in a table format. A bar chart displays the volume fractions for the current test case. *)

```
unit addunit;

interface

uses
    Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
    StdCtrls, Buttons, ExtCtrls, ComCtrls, Grids, TeEngine,
    Series, TeeProcs, Chart, Db, DBGrids ,Tomo;

type
    TResultTable = (capacitance, resistance);
    TAddToDatabase = class(TForm)
        AddReadings: TPanel;
        AddStatusBar: TStatusBar;
        AddCapTable: TStringGrid;
        AddResTable: TStringGrid;
        Label3: TLabel;
        Label4: TLabel;
        StatusUpdate: TTimer;
        Panel1: TPanel;
        Panel2: TPanel;
```

```
Label1: TLabel;
Label2: TLabel;
Label5: TLabel;
AddVolume: TChart;
Series1: TBarSeries;
Series2: TBarSeries;
Series3: TBarSeries;
AddSystemSelect: TRadioGroup;
AddReset: TSpeedButton;
AddPrevious: TSpeedButton;
AddSampleStart: TSpeedButton;
AddNext: TSpeedButton;
AddExit: TSpeedButton;
AddSetAir: TEdit;
AddSetGravel: TEdit;
AddSetWater: TEdit;
AddUserSetVolume: TCheckBox;
AddInputTomo: TTomo;
procedure AddExitClick(Sender: TObject);
procedure FormCreate(Sender: TObject);
procedure StatusUpdateTimer(Sender: TObject);
procedure FormShow(Sender: TObject);
procedure AddResetClick(Sender: TObject);
procedure AddSampleStartClick(Sender: TObject);
procedure AddPreviousClick(Sender: TObject);
procedure AddNextClick(Sender: TObject);
private
    { Private declarations }
    currenttest : integer; // current test number
    procedure ProcessResults; // process the results from the sampling hardware
    procedure UpdateTables; // update the tables with the new data
    procedure DisplayData; // display the desired output for a specific
        // test case using the InputTomo component
public
    { Public declarations }
    procedure WndProc(var TheMsg: TMessage); override;
        // override the default Windows messaging
        // procedure so as to catch the stop sampling
        // notification from the sampling hardware

end;

var
    AddToDatabase: TAddToDatabase;

implementation

uses mainunit;
{$R *.DFM}

(* ----- SAMPLING PROCEDURE ----- *)
(* process the results by extracting the data for each system (either the upper
or lower system) and storing the voltages in InpTable. Also store the volume
fractions of the different components and whether the data is for the upper or
lower system. Finally, store the picture of the desired network output in
OutTable *)
procedure TAddToDatabase.ProcessResults;
var row,col,index,vers : integer;
    s : string;
begin
    InpTable.FindKey([currenttest+0.001]);
    OutTable.FindKey([currenttest+0.001]);

    // for each test case, capture numframes of data - this is to improve
    // the noise performance of the neural network
    for vers := 1 to numframes do
        begin
            // put tables in Edit mode
            InpTable.Edit;
            OutTable.Edit;

            // store a record of which system we are currently using
            InpTable.FieldName('System').Value := AddSystemSelect.ItemIndex;
            // store the pixel image of the desired network output
            index := 1;
            for col := 0 to 9 do
                for row := 0 to 9 do
                    begin
                        s := 'Pixel'+inttostr(index);
                        OutTable.FieldName(s).Value :=
                            AddInputTomo.Desired[col,row];
```

```

        inc(index);
    end;
    // extract from the sampled data the readings specific to each
    // particular system, and store the results
    index := 1;
    col := (vers-1)*256;
    row := 0;
    if (AddSystemSelect.ItemIndex = 1) then
        col := col+8;
        while (col <= ((vers*256)-1)) do
            begin
                s := 'Voltage'+inttostr(index);
                InpTable.FieldByName(s).Value := result[col];
                inc(row);
                inc(index);
                if row = 8 then
                    begin
                        row := 0;
                        inc(col,9);
                    end
                else
                    inc(col);
                end;
            end;
            // store the volume fractions
            OutTable.FieldByName('WaterVolume').Value :=
AddInputTomo.WaterVolume;
            OutTable.FieldByName('GravelVolume').Value :=
AddInputTomo.GravelVolume;
            OutTable.FieldByName('AirVolume').Value := AddInputTomo.AirVolume;

            // force the database to write the changes to the tables
            InpTable.Post;
            OutTable.Post;

            // move to the next record in the database
            InpTable.Next;
            OutTable.Next;
        end;
        // display the current samples in the tables
        InpTable.FindKey([currenttest+0.001]);
        OutTable.FindKey([currenttest+0.001]);
        UpdateTables;
    end;

    (* start the sampling as a background process, specifying the number of
    frames of data to capture *)
    procedure TAddToDatabase.AddSampleStartClick(Sender: TObject);
    begin
        Cursor := crHourGlass;
        SampHardware.StartSample(numframes);
    end;

    (* ----- GENERAL PROCEDURES ----- *)
    (* disable the status update timer and exit *)
    procedure TAddToDatabase.AddExitClick(Sender: TObject);
    begin
        StatusUpdate.Enabled := false;
        Close;
    end;

    (* when create form, initialise the tables and the chart displaying the current
    volume fractions *)
    procedure TAddToDatabase.FormCreate(Sender: TObject);
    begin
        AddResTable.Cells[0,1] := 'TxA';
        AddResTable.Cells[0,2] := 'TxB';
        AddResTable.Cells[0,3] := 'TxC';
        AddResTable.Cells[0,4] := 'TxD';
        AddResTable.Cells[0,5] := 'TxE';
        AddResTable.Cells[0,6] := 'TxF';
        AddResTable.Cells[0,7] := 'TxG';
        AddResTable.Cells[0,8] := 'TxH';
        AddResTable.Cells[1,0] := 'RxA';
        AddResTable.Cells[2,0] := 'RxB';
        AddResTable.Cells[3,0] := 'RxC';
        AddResTable.Cells[4,0] := 'RxD';
        AddResTable.Cells[5,0] := 'RxE';
        AddResTable.Cells[6,0] := 'RxF';
        AddResTable.Cells[7,0] := 'RxG';
        AddResTable.Cells[8,0] := 'RxH';

```

```

        AddCapTable.Cells[0,1] := 'TxA';
        AddCapTable.Cells[0,2] := 'TxB';
        AddCapTable.Cells[0,3] := 'TxC';
        AddCapTable.Cells[0,4] := 'TxD';
        AddCapTable.Cells[0,5] := 'TxE';
        AddCapTable.Cells[0,6] := 'TxF';
        AddCapTable.Cells[0,7] := 'TxG';
        AddCapTable.Cells[0,8] := 'TxH';
        AddCapTable.Cells[1,0] := 'RxA';
        AddCapTable.Cells[2,0] := 'RxB';
        AddCapTable.Cells[3,0] := 'RxC';
        AddCapTable.Cells[4,0] := 'RxD';
        AddCapTable.Cells[5,0] := 'RxE';
        AddCapTable.Cells[6,0] := 'RxF';
        AddCapTable.Cells[7,0] := 'RxG';
        AddCapTable.Cells[8,0] := 'RxH';
        AddVolume.SeriesList[0].Add(AddInputTomo.WaterVolume);
        AddVolume.SeriesList[1].Add(AddInputTomo.AirVolume);
        AddVolume.SeriesList[2].Add(AddInputTomo.GravelVolume);

    end;

    (* every 10ms, update the status bar to indicate the current program situation,
    specifically, the X and Y position of the cursor on the InputTomo component,
    the status of the sampling hardware, the currenttest and totalsamples indicator
    and whether the left and right buttons are enabled (to control the browsing
    through the database or create a new test sample). Also update the chart of
    the volume fractions *)
    procedure TAddToDatabase.StatusUpdateTimer(Sender: TObject);
    begin
        AddStatusBar.Panels[0].Text := 'X: '+inttostr(AddInputTomo.Xposition);
        AddStatusBar.Panels[1].Text := 'Y: '+inttostr(AddInputTomo.Yposition);
        if AddUserSetVolume.Checked then
            begin
                AddSetGravel.Enabled := true;
                AddSetAir.Enabled := true;
                AddVolume.SeriesList[1].YValue[0] := strttoFloat(AddSetAir.Text);
                AddVolume.SeriesList[2].YValue[0] := strttoFloat(AddSetGravel.Text);
                AddVolume.SeriesList[0].YValue[0] := 100 - AddVolume.SeriesList[1].YValue[0]
                    -
                    AddVolume.SeriesList[2].YValue[0];
            end
        else
            begin
                AddSetGravel.Enabled := false;
                AddSetAir.Enabled := false;
                AddVolume.SeriesList[0].YValue[0] := AddInputTomo.WaterVolume;
                AddVolume.SeriesList[1].YValue[0] := AddInputTomo.AirVolume;
                AddVolume.SeriesList[2].YValue[0] := AddInputTomo.GravelVolume;
                AddSetWater.Text := floattostr(AddInputTomo.WaterVolume);
                AddSetAir.Text := floattostr(AddInputTomo.AirVolume);
                AddSetGravel.Text := floattostr(AddInputTomo.GravelVolume);
            end;
        AddStatusBar.Panels[3].Text := 'Current test: '+floattostr(currenttest);
        AddStatusBar.Panels[4].Text := 'Total number of samples: '+
            inttostr(InpTable.RecordCount-1);

        if currenttest = 1 then
            AddPrevious.Enabled := false
        else
            AddPrevious.Enabled := true;

        if AddInputTomo.AllowEdit = false then
            begin
                if currenttest = ((InpTable.RecordCount-1)div numframes) then
                    AddNext.Enabled := false
                else
                    AddNext.Enabled := true;
            end
        else
            AddNext.Enabled := true;

        if SampHardware.GetSampleStatus then
            AddStatusBar.Panels[2].Text := 'SAMPLING'
        else
            AddStatusBar.Panels[2].Text := 'READY';
    end;

    (* override the message handling procedure to determine whether the sampling
    hardware has completed. If it is the stop notification, then stop the hardware

```

```

and process the results, else handle the message in the default way *)
procedure TAddToDatabase.WndProc(var TheMsg:TMessage);
begin
    if TheMsg.Msg = notifvalue then
    begin
        Cursor := crDefault;
        SampHardware.StopSample(result);
        ProcessResults;
    end
    else
        inherited WndProc(TheMsg);
end;

(* when open the form, move to the last test case in the database (i.e. the
last test taken before the form was closed) and display the voltage readings in
the table. Also display what the desired output for that particular test case
was *)
procedure TAddToDatabase.FormShow(Sender: TObject);
begin
    InpTable.Last;
    OutTable.Last;
    // get the total number of test cases by moving to the last element
    // in the database and extracting the test number
    currenttest := integer(InpTable.FieldByName('TestNo').Value);
    AddInputTomo.ResetDesired;
    if currenttest <> 0 then
    begin
        InpTable.FindKey([currenttest+0.001]);
        OutTable.FindKey([currenttest+0.001]);
        // display the desired output and update the tables
        DisplayData;
        UpdateTables;
    end;
    StatusUpdate.Enabled := true;
end;

(* reset the InputTomo component *)
procedure TAddToDatabase.AddResetClick(Sender: TObject);
begin
    AddInputTomo.ResetDesired;
    AddInputTomo.Repaint;
end;

(* when click the left arrow, move to the previous test case in the database
and display the results in the tables and the desired output using the InputTomo
component *)
procedure TAddToDatabase.AddPreviousClick(Sender: TObject);
begin
    if (currenttest > 1) then
    begin
        dec(currenttest);
        InpTable.FindKey([currenttest+0.001]);
        OutTable.FindKey([currenttest+0.001]);
        DisplayData;
        UpdateTables;
    end;
end;

(* when click the right arrow, either move to the next element in the database
or create a complete new test case, depending on the position of the cursor in
the database *)
procedure TAddToDatabase.AddNextClick(Sender: TObject);
var vers,col,row: integer;
begin
    if (currenttest <> 0)and
        (currenttest <> ((InpTable.RecordCount-1) div numframes))then
    begin
        // if cursor not at the end of the database then simply move to the
        // next element in the database and display the results
        inc(currenttest);
        InpTable.FindKey([currenttest+0.001]);
        OutTable.FindKey([currenttest+0.001]);
        DisplayData;
        UpdateTables;
    end
    else
    begin
        // if at the end then create space for numframes of records and set
        // the TestNo fields. Also clear the tables to illustrate that
        // sampling must still take place

```

```

        inc(currenttest);
        for vers := 1 to numframes do
        begin
            InpTable.Append;
            OutTable.Append;
            InpTable.FieldByName('TestNo').Value := currenttest+(vers/1000);
            OutTable.FieldByName('TestNo').Value := currenttest+(vers/1000);
            for col := 1 to 8 do
                for row := 1 to 8 do
                begin
                    AddCapTable.Cells[col,row] := '0';
                    AddResTable.Cells[col,row] := '0';
                end;
            end;
        end;
    end;
end;

(* update the tables with the values in the database *)
procedure TAddToDatabase.UpdateTables;
var row, col, index, overall : integer;
    tresult : TResultTable;
    s1,s2 : string;
begin
    index := 1;
    for overall := 1 to 16 do
    begin
        case overall of
            1 : begin
                    row := 4;    tresult := capacitance;
                end;
            2 : begin
                    row := 4;    tresult := resistance;
                end;
            3 : begin
                    row := 3;    tresult := capacitance;
                end;
            4 : begin
                    row := 3;    tresult := resistance;
                end;
            5 : begin
                    row := 2;    tresult := capacitance;
                end;
            6 : begin
                    row := 2;    tresult := resistance;
                end;
            7 : begin
                    row := 1;    tresult := capacitance;
                end;
            8 : begin
                    row := 1;    tresult := resistance;
                end;
            9 : begin
                    row := 5;    tresult := resistance;
                end;
            10: begin
                    row := 5;    tresult := capacitance;
                end;
            11: begin
                    row := 6;    tresult := resistance;
                end;
            12: begin
                    row := 6;    tresult := capacitance;
                end;
            13: begin
                    row := 7;    tresult := resistance;
                end;
            14: begin
                    row := 7;    tresult := capacitance;
                end;
            15: begin
                    row := 8;    tresult := resistance;
                end;
            16: begin
                    row := 8;    tresult := capacitance;
                end;
        end;
        end;
        for col := 1 to 8 do
        begin
            s1 := 'Voltage'+inttostr(index);
            s2 := Format('%2.5f',[double(InpTable.FieldByName(s1).Value)]);

```

```

        if tresult = capacitance then
            AddCapTable.Cells[col,row] := s2
        else
            AddResTable.Cells[col,row] := s2;
            inc(index);
        end;
    end;
end;

(* set the InputTomo component based on the database to reflect the desired
output for a specific test case *)
procedure TAddToDatabase.DisplayData;
var col,row,index : integer;
    s : string;
    tempdesired : TOutput;
begin
    index := 1;
    for col := 0 to 9 do
        for row := 0 to 9 do
            begin
                s := 'Pixel'+inttostr(index);
                tempdesired[col,row] := OutTable.FieldByName(s).Value;
                inc(index);
            end;
            AddInputTomo.Desired := tempdesired;
            AddInputTomo.WaterVolume := OutTable.FieldByName('WaterVolume').Value;
            AddInputTomo.GravelVolume := OutTable.FieldByName('GravelVolume').Value;
            AddInputTomo.AirVolume := OutTable.FieldByName('AirVolume').Value;
            AddSystemSelect.ItemIndex := InpTable.FieldByName('System').Value;
            AddInputTomo.Repaint;
        end;
    end;
end.

```

PROGRAM LISTING OF CURUNIT.PAS

(* This unit continuously samples the impedance tomography system using streaming and displays the results in a tabular format. The user can specify whether to sample from the upper or lower system. It also provides the feature to automatically save frames of data to a text file at a specified rate so that the drift of the voltages can be analysed. *)

```

unit curunit;

interface

uses
    Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
    StdCtrls, ExtCtrls, Grids, Buttons, TeEngine, Series, TeeProcs, Chart,
    Spin;

type
    TCurMain = class(TForm)
        Panel1: TPanel;
        Panel2: TPanel;
        Label5: TLabel;
        Label3: TLabel;
        CurCapTable: TStringGrid;
        CurResTable: TStringGrid;
        Label4: TLabel;
        CurSystemSelect: TRadioGroup;
        CurTimer: TTimer;
        CurExit: TSpeedButton;
        DriftTimer: TTimer;
        Calibrate: TSpeedButton;
        StartMonitor: TSpeedButton;
        DriftSave: TSaveDialog;
        Label1: TLabel;
        RecordRate: TSpinEdit;
        SampleNumber: TLabel;
        procedure CurExitClick(Sender: TObject);
        procedure FormShow(Sender: TObject);
        procedure CurTimerTimer(Sender: TObject);
        procedure DriftTimerTimer(Sender: TObject);
        procedure CalibrateClick(Sender: TObject);
        procedure StartMonitorClick(Sender: TObject);
    private
        { Private declarations }
    end;

```

```

    driftfile : textfile;           // file to where frames of data
                                    // are saved for analysis
    driftfilename : string;
    sampnumber : integer;
    tempreadings : array[1..128] of double; // array to temporarily store
                                            // the readings obtained from
                                            // the sampler
    procedure ProcessResults;       // process the results and
                                    // display in table

public
    { Public declarations }
    procedure WndProc(var TheMsg: TMessage); override;
        // override the standard message
        // handling procedure to catch
        // the notification message
        // indicating completion of
        // sampling

```

end;

```

var
    CurMain: TCurMain;

```

implementation

uses mainunit, addunit;

```
{$R *.DFM}
```

```

(* ----- GENERAL PROCEDURES ----- *)
(* this procedure is called whenever the form receives a message from Windows.
If the message indicates that sampling is complete then stop the sampling and
process the readings. If not, then handle the message in the default way. *)
procedure TCurMain.WndProc(var TheMsg:TMessage);
begin
    if TheMsg.Msg = notifvalue then
        begin
            SampHardware.StopSample(result);
            ProcessResults;
        end
    else
        inherited WndProc(TheMsg);
    end;

```

end;

```

(* disable the timer and exit *)
procedure TCurMain.CurExitClick(Sender: TObject);
begin
    DriftTimer.Enabled := false;
    CurTimer.Enabled := false;
    Close;
end;

```

```

(* setup the tables and enable the timer *)
procedure TCurMain.FormShow(Sender: TObject);
begin
    CurCapTable.Col := 1;
    CurCapTable.Row := 1;
    CurResTable.Cells[0,1] := 'TxA';
    CurResTable.Cells[0,2] := 'TxB';
    CurResTable.Cells[0,3] := 'TxC';
    CurResTable.Cells[0,4] := 'TxD';
    CurResTable.Cells[0,5] := 'TxE';
    CurResTable.Cells[0,6] := 'TxF';
    CurResTable.Cells[0,7] := 'TxG';
    CurResTable.Cells[0,8] := 'TxH';
    CurResTable.Cells[1,0] := 'RxA';
    CurResTable.Cells[2,0] := 'RxB';
    CurResTable.Cells[3,0] := 'RxC';
    CurResTable.Cells[4,0] := 'RxD';
    CurResTable.Cells[5,0] := 'RxE';
    CurResTable.Cells[6,0] := 'RxF';
    CurResTable.Cells[7,0] := 'RxG';
    CurResTable.Cells[8,0] := 'RxH';
    CurCapTable.Cells[0,1] := 'TxA';
    CurCapTable.Cells[0,2] := 'TxB';
    CurCapTable.Cells[0,3] := 'TxC';
    CurCapTable.Cells[0,4] := 'TxD';
    CurCapTable.Cells[0,5] := 'TxE';
    CurCapTable.Cells[0,6] := 'TxF';
    CurCapTable.Cells[0,7] := 'TxG';

```



```

CurCapTable.Cells[0,8] := 'TxH';
CurCapTable.Cells[1,0] := 'RxA';
CurCapTable.Cells[2,0] := 'RxB';
CurCapTable.Cells[3,0] := 'RxC';
CurCapTable.Cells[4,0] := 'RxD';
CurCapTable.Cells[5,0] := 'RxE';
CurCapTable.Cells[6,0] := 'RxF';
CurCapTable.Cells[7,0] := 'RxG';
CurCapTable.Cells[8,0] := 'RxH';
CurTimer.Enabled := true;
end;

(* ----- SAMPLING PROCEDURES ----- *)
(* each time the timer overflows, initiate data capture of 1 frame of data *)
procedure TCurMain.CurTimerTimer(Sender: TObject);
begin
    SampHardware.StartSample(1);
end;

(* sort the results from the sampunit depending on whether we are sampling from
the upper or lower system and display the results in a table *)
procedure TCurMain.ProcessResults;
var row, col, index, overall : integer;
    result : TResultTable;
    s : string;
begin
    index := 1;
    col := 0;
    row := 0;

    // data from the bottom system is offset by 8 with the order being
    // 8 from top system, 8 from bottom system, 8 from top system and so on
    if (CurSystemSelect.ItemIndex = 1) then
        col := col+8;

    while (col <= 255) do
    begin
        tempreadings[index] := result[col];
        inc(row);
        inc(index);
        if row = 8 then
            begin
                row := 0;
                inc(col,9);
            end
        else
            inc(col);
        end;

    // display the results in table taking care to sort the data according
    // to the multiplexor inputs on the synchronous detector cards
    index := 1;
    for overall := 1 to 16 do
    begin
        case overall of
            1 : begin
                    row := 4;    tresult := capacitance;
                end;
            2 : begin
                    row := 4;    tresult := resistance;
                end;
            3 : begin
                    row := 3;    tresult := capacitance;
                end;
            4 : begin
                    row := 3;    tresult := resistance;
                end;
            5 : begin
                    row := 2;    tresult := capacitance;
                end;
            6 : begin
                    row := 2;    tresult := resistance;
                end;
            7 : begin
                    row := 1;    tresult := capacitance;
                end;
            8 : begin
                    row := 1;    tresult := resistance;
                end;
            9 : begin
                    row := 5;    tresult := resistance;
                end;
        end;
    end;

    end;

10: begin
    row := 5;    tresult := capacitance;
end;
11: begin
    row := 6;    tresult := resistance;
end;
12: begin
    row := 6;    tresult := capacitance;
end;
13: begin
    row := 7;    tresult := resistance;
end;
14: begin
    row := 7;    tresult := capacitance;
end;
15: begin
    row := 8;    tresult := resistance;
end;
16: begin
    row := 8;    tresult := capacitance;
end;
end;
for col := 1 to 8 do
begin
    s := Format('%2.3f',[double(tempreadings[index]))];
    if tresult = capacitance then
        CurCapTable.Cells[col,row] := s
    else
        CurResTable.Cells[col,row] := s;
    inc(index);
end;
end;

end;

// ----- VOLTAGE MONITORING PROCEDURES ----- //
// each time the drift timer overflows, open the text file, add all 128 voltages
// as a single line to the file (separated by spaces) and close the file again-
// also increment sampnumber to give visual indication of the number of frames
// that have been saved to this file
procedure TCurMain.DriftTimerTimer(Sender: TObject);
var value : string;
    row, col : integer;
begin
    value := '';
    for col := 1 to 8 do
        for row := 1 to 8 do
            value := value + CurCapTable.Cells[col,row]+' ';
        end;
        for col := 1 to 8 do
            for row := 1 to 8 do
                value := value + CurResTable.Cells[col,row]+' ';
            end;
        end;
        AssignFile(driftfile, driftfilename);
        Append(driftfile);
        WriteLn(driftfile,value);
        CloseFile(driftfile);
        SampleNumber.Caption := inttostr(sampnumber);
        inc(sampnumber);
    end;

// procedure to write the contents of tables to Excel spreadsheet for analysis
// at a later stage
procedure TCurMain.CalibrateClick(Sender: TObject);
var tempfile : textfile;
    row, col : integer;
begin
    if DriftSave.Execute then
    begin
        AssignFile(tempfile,DriftSave.FileName);
        ReWrite(tempfile);
        if CurSystemSelect.ItemIndex = 0 then
            WriteLn(tempfile, 'Top system')
        else
            WriteLn(tempfile, 'Bottom system');
        WriteLn(tempfile, DateToStr(Date));
        WriteLn(tempfile, 'Capacitance');
        for row := 0 to 8 do
            begin

```

```

        for col := 0 to 8 do
            Write(tempfile, CurCapTable.Cells[col,row]+' ');
            WriteLn(tempfile, ' ');
        end;
        WriteLn(tempfile, 'Resistance');
        for row := 0 to 8 do
            begin
                for col := 0 to 8 do
                    Write(tempfile, CurResTable.Cells[col,row]+' ');
                    WriteLn(tempfile, ' ');
                end;
            end;
        end;
        CloseFile(tempfile);
    end;
end;

// start drift analysis by specifying destination filename and desired
// capture rate
procedure TCurMain.StartMonitorClick(Sender: TObject);
begin
    if DriftTimer.Enabled = false then
        begin
            if DriftSave.Execute then
                begin
                    sampnumber := 1;
                    SampleNumber.Visible := true;
                    AssignFile(driftfile, DriftSave.FileName);
                    driftfilename := DriftSave.FileName;
                    ReWrite(driftfile);
                    StartMonitor.Caption := 'Stop recording';
                    DriftTimer.Interval := Round(3.6e6/RecordRate.Value);
                    DriftTimer.Enabled := true;
                    CloseFile(driftfile);
                end;
            end
        else
            begin
                StartMonitor.Caption := 'Start recording';
                DriftTimer.Enabled := false;
                SampleNumber.Visible := false;
            end;
        end;
    end;
end.

```

PROGRAM LISTING OF SAMPUNIT.PAS

(* This unit controls the sampling of the Eagle PC30G card. The sampling is controlled by the sample control board which sets up the multiplexers and triggers the card to start sampling. The card is configured for burst mode and samples channel 0 to 15 each time it receives a trigger on the external trigger line. Streaming is used to capture the samples as a background process- it is configured using the EDR32 unit provided with the card. Once the sampling is complete, a Windows notification message is issued by EDR32. This message is captured by the sampling software and the data is then processed. For the sampling to work correctly, the PC30G card must be configured as follows:

A/D configuration: single ended
 -5V to +5V range
 slave trigger

DMA: single DMA channel 5
 base address: 1000H *)

unit sampunit;

interface

uses EDR32;

const rst555 : byte = \$00; // sequence to reset 555 on sample control card
 rstclk : byte = \$02; // sequence to reset counter on sample control card
 std555 : byte = \$01; // sequence for run mode
 datamask : word = \$0FFF; // mask to extract A/D reading from raw binary data

type

TSamp = class
 private
 boardnum : integer;
 numframes : integer; // number of frames to capture
 busy_sampling : Boolean; // Boolean indicating whether

```

        // the system is currently
        // sampling
        buffer : array[0..65535] of word; // buffer store for sampled data
    public
        // initialise the card and get Windows notification value
        function InitSample(bdnum: integer; Handle: cardinal): cardinal;
        // check the status of the sampling
        function GetSampleStatus: Boolean;
        // release the resources allocated to sampling
        procedure ReleaseSample;
        // start sampling where the total number of frames to be
        // captured is nmframes
        procedure StartSample(nmframes: integer);
        // once sampling is complete, process sampled data
        procedure StopSample(var result: array of double);
    end;

```

implementation

```

// initialise the sampling by getting the notification message value, setting
// digital IO port A to output, loading channel list with channel 0 to 15,
// setting the burst length to 16, setting the transfer mode to streaming
// and the minimum block size to 1024
function TSamp.InitSample(bdnum: integer; Handle: cardinal): cardinal;
var message_val : cardinal;
    i : integer;
    frequency : integer;
begin
    frequency := 6000;
    boardnum := bdnum;
    // get the message number that is sent to the notification window
    message_val := EDR_GetNotificationMsg;
    // set the window that will receive the notification messages
    // for the board
    EDR_SetNotificationWindow(boardnum, Handle);
    numframes := 0;
    // configure port A as output port in simple DIO mode
    EDR_DIOConfigurePort(boardnum, 0, EDR_DIO_SIMPLE, 0);
    // reset 555 and counter on sample control card
    EDR_DIOPortOutput(boardnum, 0, rst555);
    EDR_DIOPortOutput(boardnum, 0, rstclk);
    // load the channel list with channel 0 to 15
    EDR_SetADChanListLen(boardnum, 0);
    for i:=0 to 15 do begin
        EDR_AddToADChanList(boardnum, i);
        EDR_SetADInGain(boardnum, i, 1);
    end;
    EDR_SetADClockmilliHz(boardnum, frequency*1000);
    // set the transfer mode of the AD data as streaming
    EDR_SetADTransferMode(boardnum, EDR_STREAM);
    EDR_SetADBurstLen(boardnum, EDR_ADBURSTNUMCHANS);
    // set the size of the block written out by the streamer
    EDR_SetStreamBlockSize(boardnum, 1024);
    InitSample := message_val;
    busy_sampling := false;
end;

// release the resources allocated for sampling
procedure TSamp.ReleaseSample;
begin
    EDR_DIOPortOutput(boardnum, 0, 0);
    EDR_SetNotificationWindow(boardnum, 0);
    EDR_StopBackgroundADIn(boardnum);
    EDR_FreeBoardHandle(boardnum);
end;

// start sampling by resetting the 555 and counter, instructing the EDR
// software to capture a specified number of samples and then putting
// the 555 in run mode
procedure TSamp.StartSample(nmframes: integer);
var i : integer;
begin
    busy_sampling := true;
    numframes := nmframes;
    // reset 555 and counter
    EDR_DIOPortOutput(boardnum, 0, rst555);
    EDR_DIOPortOutput(boardnum, 0, rstclk);
    EDR_DIOPortOutput(boardnum, 0, rst555);
    // acquire a block of AD data using streaming and place the data
    // into buffer

```

```

EDR_ADInBinBackground(boardnum,256*numframes,buffer[0]);
// create a delay while the Eagle card is configured
for i := 1 to 1000 do
    EDR_DIOPortOutput(boardnum,0,rst555);
// start the sampling by putting the 555 in run mode
EDR_DIOPortOutput(boardnum,0,std555);
end;

// stop the sampling by calling EDR_StopBackgroundADIn, resetting the 555
// and counter and process raw binary data into voltages in the range from
// -5V to +5V - these voltages are then stored in the 'result' array
procedure TSamp.StopSample(var result: array of double);
var temp : double;
    i : integer;
begin
    EDR_StopBackgroundADIn(boardnum);
    EDR_DIOPortOutput(boardnum,0,rst555);
    EDR_DIOPortOutput(boardnum,0,rstclk);
    EDR_DIOPortOutput(boardnum,0,rst555);
    for i := 0 to (256*numframes - 1) do
        begin
            temp := buffer[i] AND datamask;
            temp := (temp-2048)*5/2048;
            result[i] := temp;
        end;
    busy_sampling := false;
end;

// get status of the sampling
function TSamp.GetSampleStatus;
begin
    GetSampleStatus := busy_sampling;
end;

end.

```

PROGRAM LISTING OF TOMO.PAS

(* This unit implements a tomography component which provides a convenient way to display the contents of a pipeline as well as a way to specify the contents of a pipeline for training database generation. The component segments the pipeline into a 10 by 10 matrix of pixels where the phase of each pixel can be either water, gravel or air. Of the 100 pixels, only 88 fall within the area of the pipe and the rest are classified as outpipe. *)

```

unit Tomo;

interface

uses
    Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
    ExtCtrls;

type
    TBubblePhase = (air,water,gravel,outpipe); // dedicated type to identify
                                                // the phase of a pixel
    TOutput = array [0..9,0..9] of TBubblePhase; // dedicated type to store the
                                                // 10 by 10 matrix of pixels,
                                                // where each pixel is of type
                                                // TBubblePhase
    TTomo = class(TPaintBox) // component inherits from the
                            // standard Delphi paintbox
    private
        { Private declarations }
        FAllowEdit : Boolean;
        FWaterVolume : Integer;
        FAirVolume : Integer;
        FGravelVolume : Integer;
        FXPosition : Integer;
        FYPosition : Integer;
        FDesired : TOutput;
        procedure SetDesired(row:integer;col:integer; value:TBubblePhase);
    protected
        { Protected declarations }
        procedure MouseDown(Sender: TObject; Button: TMouseButton; Shift:
            TShiftState; X, Y: Integer);virtual;
        procedure MouseMove(Sender: TObject; Shift: TShiftState; X,Y: Integer);virtual;
        procedure Paint;override;

```

```

public
    { Public declarations }
    constructor Create(AOwner : TComponent); override; // create tomo component
    procedure ResetDesired; // reset all the pixels to water
    property Desired : TOutput
        read FDesired write FDesired;

published
    { Published declarations }
    // property specifying whether the data can be changed or not
    property AllowEdit: Boolean
        read FAllowEdit write FAllowEdit default true;
    // property specifying the percentage water in the pipeline
    property WaterVolume : Integer
        read FWaterVolume write FWaterVolume default 100;
    // property specifying the percentage gravel in the pipeline
    property GravelVolume : Integer
        read FGravelVolume write FGravelVolume default 0;
    // property specifying the percentage air in the pipeline
    property AirVolume : Integer
        read FAirVolume write FAirVolume default 0;
    // current X and Y positions of the cursor
    property XPosition : Integer
        read FXPosition;
    property YPosition : Integer
        read FYPosition;
    property OnPaint;

end;

procedure Register;

implementation

// register the component and place the icon on the 'samples' palette
procedure Register;
begin
    RegisterComponents('Samples', [TTomo]);
end;

{ TTomo }

// create a paintbox of specified size and initialise the volume fractions
// for 100% water
constructor TTomo.Create(AOwner: TComponent);
begin
    inherited;
    Width := 200;
    Height := 200;
    FWaterVolume := 100;
    FGravelVolume := 0;
    FAirVolume := 0;
    ResetDesired;
    Self.OnMouseDown := MouseDown;
    Self.OnMouseMove := MouseMove;
end;

// the phase of each pixel is set by clicking on the pixel as follows
// left mouse button - air
// central button - water
// right mouse button - gravel
// however, only change the pixel phase if the cursor is inside the pipe
// and if editing of the pipeline contents is enabled
procedure TTomo.MouseDown(Sender: TObject; Button: TMouseButton;
    Shift: TShiftState; X, Y: Integer);
var row,col,newX,newY : integer;
begin
    row := X div 20;
    col := Y div 20;
    if (FDesired[row,col] <> outpipe)and(FAllowEdit) then
        begin
            // air pixel
            if (ssLeft in Shift) then
                begin
                    Canvas.Pen.Color := clWhite;
                    Canvas.Brush.Color := clWhite;
                    SetDesired(row,col,air);
                end
            // gravel pixel
            else if (ssRight in Shift) then

```

```

begin
    Canvas.Pen.Color := clGray;
    Canvas.Brush.Color := clGray;
    SetDesired(row,col,gravel);
end
// water pixel
else
begin
    Canvas.Pen.Color := clBlue;
    Canvas.Brush.Color := clBlue;
    SetDesired(row,col,water);
end;
newX := row*20;
newY := col*20;
Canvas.Rectangle(newX,newY,newX+20,newY+20);
end;

end;

// the pipeline cross-section is painted as a sequence of squares where the
// colour of each square is specified by the 'Desired' property
procedure TTomo.Paint;
var row,col,X,Y : integer;
begin
    for row := 0 to 9 do
        begin
            for col := 0 to 9 do
                begin
                    X := (row*20);
                    Y := (col*20);
                    if FDesired[row,col] = water then
                        begin
                            Canvas.Brush.Color := clBlue;
                            Canvas.Pen.Color := clBlue;
                        end
                    else if FDesired[row,col] = gravel then
                        begin
                            Canvas.Brush.Color := clGray;
                            Canvas.Pen.Color := clGray;
                        end
                    else if FDesired[row,col] = air then
                        begin
                            Canvas.Brush.Color := clWhite;
                            Canvas.Pen.Color := clWhite;
                        end
                    else
                        begin
                            Canvas.Brush.Color := clBtnFace;
                            Canvas.Pen.Color := clBtnFace;
                        end;
                    Canvas.Rectangle(X,Y,X+20,Y+20);
                end;
            end;
        end;
    end;

    // reset all the pixels in the pipeline to water
    procedure TTomo.ResetDesired;
    var row,col,bottom,top : integer;
    begin
        FWaterVolume := 100;
        FGravelVolume := 0;
        FAirVolume := 0;

        Canvas.Pen.Color := clBlue;
        Canvas.Pen.Width := 1;
        Canvas.Brush.Color := clBlue;
        for row := 0 to 9 do
            begin
                for col := 0 to 9 do
                    begin
                        FDesired[row,col] := outpipe;
                    end;
                end;
            end;

            for row := 0 to 9 do
                begin
                    case row of
                        0: begin top := 2; bottom := 7; end;
                        1: begin top := 1; bottom := 8; end;
                        8: begin top := 1; bottom := 8; end;
                    end;
                end;
            end;
        end;
    end;

    9: begin top := 2; bottom := 7; end;
    2..7: begin top := 0; bottom := 9; end;
end;
for col := top to bottom do
begin
    FDesired[row,col] := water;
end;
end;

// adjust the volume fractions of the different phases according to the
// change in pixel at position [row,col] to phase 'value'
procedure TTomo.SetDesired(row,col: integer; value: TBubblePhase);
begin
    if (row <= 9) and (col <= 9) then
        begin
            if FDesired[row,col] <> outpipe then
                begin
                    if FDesired[row,col] = water then
                        dec(FWaterVolume)
                    else if FDesired[row,col] = gravel then
                        dec(FGravelVolume)
                    else
                        dec(FAirVolume);
                    if value = water then
                        inc(FWaterVolume)
                    else if value = gravel then
                        inc(FGravelVolume)
                    else
                        inc(FAirVolume);
                    FDesired[row,col] := value;
                end;
            end;
        end;
    end;

    // pixels can also be drawn while the mouse is moving over the pipeline
    // depending on what mouse button is held down while the mouse moves
    procedure TTomo.MouseMove(Sender: TObject; Shift: TShiftState; X,
        Y: Integer);
    var row,col,newX,newY : integer;
    begin
        row := X div 20;
        col := Y div 20;

        FXPosition := row+1;
        FYPosition := col+1;
        if ((ssLeft in Shift) or (ssRight in Shift) or (ssMiddle in Shift)) then
            begin
                if (FDesired[row,col] <> outpipe) and (FAAllowEdit) then
                    begin
                        // air phase
                        if (ssLeft in Shift) then
                            begin
                                Canvas.Pen.Color := clWhite;
                                Canvas.Brush.Color := clWhite;
                                SetDesired(row,col,air);
                            end
                        // gravel phase
                        else if (ssRight in Shift) then
                            begin
                                Canvas.Pen.Color := clGray;
                                Canvas.Brush.Color := clGray;
                                SetDesired(row,col,gravel);
                            end
                        // water phase
                        else
                            begin
                                Canvas.Pen.Color := clBlue;
                                Canvas.Brush.Color := clBlue;
                                SetDesired(row,col,water);
                            end;
                        newX := row*20;
                        newY := col*20;
                        Canvas.Rectangle(newX,newY,newX+20,newY+20);
                    end;
                end;
            end;
        end;
    end.
end.

```

APPENDIX L

NEURAL NETWORK PROGRAM CODE

PROGRAM LISTING OF MAINUNIT.PAS

(* This program will be used to train and test neural networks for the image reconstruction and volume fraction prediction of a static vessel situation. The neural networks provided are the single-layer feed-forward multi-layer perceptron and the double-layer feed-forward multi-layer perceptron. These networks can be trained using gradient descent or Resilient back-propagation - these network training algorithms have been taken from the previous network training program.

The main unit provides the general structure to the program through a menu. It also performs functions such as loading and saving the network weights, saving the network performance and initialising the network weights to random values. All data matrices are allocated dynamically once the network training parameters have been specified - this saves memory.*)

unit mainunit;

interface

uses

Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs, ExtCtrls, Menus, Registry, StdCtrls, Spin, Buttons, sampunit;

type

```
TRecMain = class(TForm)
  RecMenu: TMainMenu;
  File1: TMenuItem;
  LoadDatabase1: TMenuItem;
  N1: TMenuItem;
  LoadNetworkWeights: TMenuItem;
  SaveNetworkWeights: TMenuItem;
  N2: TMenuItem;
  SaveNetworkPerformance1: TMenuItem;
  N3: TMenuItem;
  Exit1: TMenuItem;
  NetworkTraining1: TMenuItem;
  SetTestNumber1: TMenuItem;
  InitialiseNetworkWeights1: TMenuItem;
  SetNetworkTrainingParameters1: TMenuItem;
  StartTraining1: TMenuItem;
  NetworkTesting1: TMenuItem;
  ValidateDatabase1: TMenuItem;
  ValidateCurrentSamples1: TMenuItem;
  Panel1: TPanel;
  ViewDatabase1: TMenuItem;
  WriteDataToTextFile1: TMenuItem;
  RecSetTestNumber: TPanel;
  Label1: TLabel;
  RecNewTestNumber: TSpinEdit;
  RecSetTestNumberOK: TBitBtn;
  RecSetTestNumberCan: TBitBtn;
  RecSetWeightFactor: TPanel;
  Label2: TLabel;
  RecWeightFactor: TEdit;
  RecWeightOK: TBitBtn;
  RecWeightCan: TBitBtn;
  RealTimeTesting1: TMenuItem;
  SaveNetworkPreprocessing1: TMenuItem;
  SaveCalibrationData1: TMenuItem;
  SaveCalibDialog: TSaveDialog;
  procedure Exit1Click(Sender: TObject);
  procedure LoadDatabase1Click(Sender: TObject);
  procedure ViewDatabase1Click(Sender: TObject);
  procedure WriteDataToTextFile1Click(Sender: TObject);
  procedure FormShow(Sender: TObject);
  procedure FormClose(Sender: TObject; var Action: TCloseAction);
  procedure RecSetTestNumberOKClick(Sender: TObject);
  procedure RecSetTestNumberCanClick(Sender: TObject);
  procedure SetTestNumber1Click(Sender: TObject);
  procedure SetNetworkTrainingParameters1Click(Sender: TObject);
  procedure InitialiseNetworkWeights1Click(Sender: TObject);
  procedure RecWeightCanClick(Sender: TObject);
```

```
  procedure RecWeightOKClick(Sender: TObject);
  procedure StartTraining1Click(Sender: TObject);
  procedure SaveNetworkPerformance1Click(Sender: TObject);
  procedure SaveNetworkWeightsClick(Sender: TObject);
  procedure LoadNetworkWeightsClick(Sender: TObject);
  procedure ValidateDatabase1Click(Sender: TObject);
  procedure ValidateCurrentSamples1Click(Sender: TObject);
  procedure FormCreate(Sender: TObject);
  procedure RealTimeTesting1Click(Sender: TObject);
  procedure SaveNetworkPreprocessing1Click(Sender: TObject);
  procedure SaveCalibrationData1Click(Sender: TObject);
private
  { Private declarations }
  weightfactor : real; // store factor by which randomly
                        // initialised weights are multiplied
public
  { Public declarations }
end;
```

var

```
RecMain: TRecMain;
SampHardware : TSamp; // provide interface to the
                      // sampling hardware
InputData : array of array of real; // input voltage readings
OutputData : array of array of real; // desired network outputs
OutputNetwork : array of array of real; // network predictions
OutputHidden : array of array of real; // hidden layer neuron outputs
OutputWeight : array of array of real; // weights for output layer
DiffOutputWeight : array of array of real; // delta values for the output layer
OldOutputWeight : array of array of real; // old output layer weights
HiddenWeight : array of array of real; // weights for hidden layer
DiffHiddenWeight : array of array of real; // delta values for the hidden layer
OldHiddenWeight : array of array of real; // old hidden layer weights
OutputDeltaWeight : array of array of real; // output layer delta values for RPROP
HiddenDeltaWeight : array of array of real; // hidden layer delta values for RPROP
OldDiffHiddenWeight : array of array of real; // old output delta values for RPROP
OldDiffOutputWeight : array of array of real; // old hidden delta values for RPROP
VolumeData : array of array of real; // volume fractions
OutputWeightBest : array of array of real; // best weights for output layer
// in model search
HiddenWeightBest : array of array of real; // best weights for hidden layer
// in model search
testnumber : integer; // overall test number which is
                      // automatically appended to
                      // all files saved
datacompare : array[1..128] of real;
(* the dimensions of the matrices is as follows
```

```
OutputData, OutputNetwork
  traintotal + testtotal X numoutputs
InputData
  traintotal + testtotal X 129
OutputHidden
  traintotal + testtotal X numhidden + 1
HiddenWeight, DiffHiddenWeight, OldHiddenWeight, HiddenDeltaWeight
  129 X numhidden
OutputWeight, OldOutputWeight, DiffOutputWeight, OutputDeltaWeight
  numhidden + 1 OR 129 X numoutputs
*)
```

implementation

uses loadunit, viewunit, paramunit, networkunit, testunit, realunit;

(\$R *.DFM)

(* ----- GENERAL PROCEDURES ----- *)

```
(* exit the program *)
procedure TRecMain.Exit1Click(Sender: TObject);
begin
  Close;
end;
```

(* show the load databases form *)

```
procedure TRecMain.LoadDatabase1Click(Sender: TObject);
```

```

begin
    RecLoadDBase.ShowModal;
end;

(* show the view databases form *)
procedure TRecMain.Viewdatabase1Click(Sender: TObject);
begin
    RecViewMain.ShowModal;
end;

(* when show form, load the previous test number and weight factor from the
registry *)
procedure TRecMain.FormShow(Sender: TObject);
var Reg : TRegistry;
    KeyGood: Boolean;
begin
    Reg := TRegistry.Create;
    try
        KeyGood := Reg.OpenKey('Software\Neural', False);
        // if the key exists then read from the key, else it is the
        // first time the program is running so do not attempt
        // to read from the registry
        if KeyGood then
            testnumber := Reg.ReadInteger('TestNumber');
            weightfactor := Reg.ReadFloat('WeightFactor');
        finally
            Reg.Free;
        end;
        RecNewTestNumber.Value := testnumber;
        Randomize;
    end;

    (* when close form, clear the memory allocated for the matrices and write
the testnumber and weightfactor to the registry *)
    procedure TRecMain.FormClose(Sender: TObject; var Action: TCloseAction);
    var Reg: TRegistry;
    begin
        Reg := TRegistry.Create;
        try
            Reg.OpenKey('Software\Neural', True);
            Reg.WriteInteger('TestNumber',testnumber);
            Reg.WriteFloat('WeightFactor',weightfactor);
        finally
            Reg.Free;
        end;

        // free dynamically allocated memory
        InputData := nil;
        OutputData := nil;
        VolumeData := nil;
        OutputNetwork := nil;
        OutputHidden := nil;
        OutputWeight := nil;
        DiffOutputWeight := nil;
        OldOutputWeight := nil;
        HiddenWeight := nil;
        DiffHiddenWeight := nil;
        OldHiddenWeight := nil;
        OutputDeltaWeight := nil;
        HiddenDeltaWeight := nil;
        OldDiffHiddenWeight := nil;
        OldDiffOutputWeight := nil;
        OutputWeightBest := nil;
        HiddenWeightBest := nil;

        // free the resources allocated for the sampling hardware
        SampHardware.ReleaseSample;
        SampHardware.Free;
    end;

    (* set the test number *)
    procedure TRecMain.RecSetTestNumberOKClick(Sender: TObject);
    begin
        testnumber := RecNewTestNumber.Value;
        RecSetTestNumber.Visible := false;
    end;

    (* user decides not to set the test number *)
    procedure TRecMain.RecSetTestNumberCanClick(Sender: TObject);
    begin
        RecSetTestNumber.Visible := false;

```

```

end;

(* allow the user to set the test number *)
procedure TRecMain.Settestnumber1Click(Sender: TObject);
begin
    RecSetTestNumber.Visible := true;
end;

(* show the form to set the network training parameters *)
procedure TRecMain.Setnetworktrainingparameters1Click(Sender: TObject);
begin
    RecSetParam.ShowModal;
end;

(* show the form to start the network training *)
procedure TRecMain.Starttraining1Click(Sender: TObject);
begin
    RecTrainNetwork.ShowModal;
end;

(* show the form to test the network performance on the testing database *)
procedure TRecMain.Validatedatabase1Click(Sender: TObject);
begin
    usecurrentsamples := false;
    RecTest.ShowModal;
end;

(* show the form to test the network performance on the current samples *)
procedure TRecMain.Validatecurrentsamples1Click(Sender: TObject);
begin
    usecurrentsamples := true;
    RecTest.ShowModal;
end;

(* create the object to access the sampling hardware *)
procedure TRecMain.FormCreate(Sender: TObject);
begin
    SampHardware := TSamp.Create;
end;

(* show real time testing form - this is where data is captured from both
systems simultaneously and reconstructions are performed in real time with
a cross-correlation algorithm calculating the velocities of the individual
components on completion *)
procedure TRecMain.Realtimetesting1Click(Sender: TObject);
begin
    RealMain.ShowModal;
end;

(* ----- FILE HANDLING PROCEDURES ----- *)
(* all file saves and loads are from a directory called 'TestData' *)
(* write the loaded data to a text file so that it can be used in Matlab *)
procedure TRecMain.Writedataatextfile1Click(Sender: TObject);
var tempfile:textfile;
    filename : string;
    row, col, endcol : integer;
begin
    endcol := RecLoadDBase.GetNumOutputs;

    // use the test number to specify the filename
    filename := 'c:\TestData\mlbtst'+inttostr(testnumber)+'\'.txt';
    AssignFile(tempfile,filename);
    Rewrite(tempfile);
    WriteLn(tempfile, RecLoadDBase.GetTotalTrain);
    // write the input voltages for the training data to the text file
    for row := 0 to RecLoadDBase.GetTotalTrain -1 do
        begin
            for col := 1 to 128 do
                Write(tempfile,InputData[row,col]);
                WriteLn(tempfile);
            end;
            WriteLn(tempfile,endcol);
        end;
    // write the desired output for the training data to the text file
    for row := 0 to RecLoadDBase.GetTotalTrain -1 do
        begin
            for col := 0 to endcol - 1 do
                Write(tempfile,OutputData[row,col]);
                WriteLn(tempfile);
            end;
        end;
    end;
end;

```

```

WriteLn(tempfile, RecLoadDBase.GetTotalTest);
// write the input voltages for the testing data to the text file
for row := RecLoadDBase.GetTotalTrain to RecLoadDBase.GetTotalTrain
+ RecLoadDBase.GetTotalTest -1 do
begin
  for col := 1 to 128 do
    Write(tempfile, InputData[row,col]);
  WriteLn(tempfile);
end;
WriteLn(tempfile, endcol);
// write the desired output for the testing data to the text file;
for row := RecLoadDBase.GetTotalTrain to RecLoadDBase.GetTotalTrain
+ RecLoadDBase.GetTotalTest -1 do
begin
  for col := 0 to endcol - 1 do
    Write(tempfile, OutputData[row,col]);
  WriteLn(tempfile);
end;

CloseFile(tempfile);
end;

(* save all the network parameters describing the current test, as well as the
network training results, to a text file so that they can be viewed at a later
stage *)
procedure TRecMain.Savenetworkperformance1Click(Sender: TObject);
var tempfile: textfile;
    filename, s: string;
begin
  // the test number is used to specify the filename
  filename := 'c:\TestData\perf'+inttostr(testnumber)+'\txt';
  AssignFile(tempfile, filename);
  ReWrite(tempfile);

  // write the training parameters to the file - this will allow repeating
  // the test at a later stage since all the required parameters are saved
  WriteLn(tempfile, 'Test number:'+inttostr(testnumber));
  WriteLn(tempfile, 'Number of outputs:'+inttostr(RecLoadDBase.GetNumOutputs));
  WriteLn(tempfile, 'Training database:'+RecLoadDBase.RecTrainName.Text);
  WriteLn(tempfile, 'Test database:'+RecLoadDBase.RecTestName.Text);
  WriteLn(tempfile, 'Train start:'+inttostr(RecLoadDBase.RecTrainStart.Value));
  WriteLn(tempfile, 'Train stop:'+inttostr(RecLoadDBase.RecTrainStop.Value));
  WriteLn(tempfile, 'Test start:'+inttostr(RecLoadDBase.RecTestStart.Value));
  WriteLn(tempfile, 'Test stop:'+inttostr(RecLoadDBase.RecTestStop.Value));
  WriteLn(tempfile, 'Max epochs:'+inttostr(RecSetParam.GetMaxEpochs));
  WriteLn(tempfile, 'Number of hidden
nodes:'+inttostr(RecSetParam.GetNumHidden));
  if RecSetParam.RecPerformEarlyStop.Checked then
    s := 'true'
  else
    s := 'false';
  WriteLn(tempfile, 'Perform early stopping:'+s);
  if RecSetParam.RecPerformModel.Checked then
    s := 'true'
  else
    s := 'false';
  WriteLn(tempfile, 'Conduct model search:'+s);
  if RecSetParam.RecUseGradDescent.Checked then
    s := 'Gradient descent'
  else
    s := 'RPROP';
  WriteLn(tempfile, 'Training algorithm:'+s);
  WriteLn(tempfile, 'Learning rate:'+floattostr(RecSetParam.GetLearnRate));
  WriteLn(tempfile, 'Momentum:'+floattostr(RecSetParam.GetMomentum));
  WriteLn(tempfile, 'Initial delta:'+floattostr(RecSetParam.GetDeltaInit));
  WriteLn(tempfile, 'Maximum delta:'+floattostr(RecSetParam.GetDeltaMax));
  WriteLn(tempfile, 'Update frequency:'+floattostr(RecSetParam.GetUpdateFreq));
  WriteLn(tempfile, 'Number of fails:'+floattostr(RecSetParam.GetNumFails));
  if RecSetParam.RecStopVolume.Checked then
    s := 'Sum volume'
  else
    s := 'Threshold';
  WriteLn(tempfile, 'Stopping condition:'+s);
  WriteLn(tempfile, 'Minimum hidden for model:'
+inttostr(RecSetParam.RecModelStart.Value));
  WriteLn(tempfile, 'Maximum hidden for model:'
+inttostr(RecSetParam.RecModelStop.Value));
  WriteLn(tempfile, 'Increment for model:'
+inttostr(RecSetParam.RecModelInc.Value));

  WriteLn(tempfile, 'Weight factor:'+floattostr(weightfactor));

  // save the network performance results to the file
  WriteLn(tempfile, 'Threshold error:'
+floattostr(RecTrainNetwork.GetThresholdError));
  WriteLn(tempfile, 'Sum void error:'
+floattostr(RecTrainNetwork.GetSumVoidError));
  WriteLn(tempfile, 'Water error:'+floattostr(RecTrainNetwork.GetWaterError));
  WriteLn(tempfile, 'Gravel error:'+floattostr(RecTrainNetwork.GetGravelError));
  WriteLn(tempfile, 'Air error:'+floattostr(RecTrainNetwork.GetAirError));
  WriteLn(tempfile, 'Number of hidden for best:'
+inttostr(RecTrainNetwork.GetNumBest));
  CloseFile(tempfile);
end;

(* save the data used to pre-process the training data to a text file so that
future tests can be done without having to load the training database - real
time captured data has to be pre-processed according to the same rules as the
training data that was used to produce the neural network being used for the
real time test *)
procedure TRecMain.Savenetworkpreprocessing1Click(Sender: TObject);
var tempfile : textfile;
    filename : string;
    tempair, tempgrav : Boolean;
    i : integer;
begin
  filename := 'c:\TestData\preproc'+inttostr(testnumber)+'\txt';
  AssignFile(tempfile, filename);
  ReWrite(tempfile);
  // keep a record of the mean of each input voltage
  for i := 1 to 128 do
    WriteLn(tempfile, meanrec[i]);
  // keep a record of the standard deviation of each input voltage
  for i := 1 to 128 do
    WriteLn(tempfile, stddevrec[i]);
  if RecLoadDBase.GetNumOutputs = 88 then
    begin
      RecLoadDBase.GetPhases(tempair, tempgrav);
      if tempair then
        WriteLn(tempfile, '0')
      else
        WriteLn(tempfile, '1');
    end;
  CloseFile(tempfile);
end;

(* save the calibration data to a comma separated textfile for analysis at a
later stage *)
procedure TRecMain.Savecalibrationdata1Click(Sender: TObject);
var tempfile : textfile;
    index, overall, col, row : integer;
    outstring : string;
    tresult : TResultTable;
    tempcappable : array[1..8, 1..8] of double;
    temprestable : array[1..8, 1..8] of double;
begin
  row := 0;
  tresult := capacitance;

  if SaveCalibDialog.Execute then
    begin
      AssignFile(tempfile, SaveCalibDialog.FileName);
      ReWrite(tempfile);
      // to save the data in a representative format, create a
      // temporary table and then sort the data accordingly
      index := 1;
      for overall := 1 to 16 do
        begin
          case overall of
            1 : begin
              row := 4;    tresult := capacitance;
            end;
            2 : begin
              row := 4;    tresult := resistance;
            end;
            3 : begin
              row := 3;    tresult := capacitance;
            end;
            4 : begin

```



```

end;
RecSetWeightFactor.Visible := false;
Starttraining1.Enabled := true;
end;

(* save the network weights to a textfile - if a model search has been done then
save the weights for the best network *)
procedure TRecMain.SavenetworkweightsClick(Sender: TObject);
var tempfile : textfile;
    filename : string;
    rowindex,colindex,upperindex : integer;
begin
    upperindex := 128;
    filename := 'c:\TestData\weight'+inttostr(testnumber)+'\txt';
    AssignFile(tempfile,filename);
    Rewrite(tempfile);
    WriteLn(tempfile,RecLoadDBase.GetNumOutputs);
    if RecSetParam.RecPerformModel.Checked then
    begin
        // if a model search has been done then write the weights for the
        // best network
        upperindex := RecTrainNetwork.GetNumBest;
        WriteLn(tempfile,upperindex);
        for colindex := 0 to upperindex - 1 do
            for rowindex := 0 to 128 do
                WriteLn(tempfile,HiddenWeightBest[rowindex,colindex]);
            for colindex := 0 to RecLoadDBase.GetNumOutputs - 1 do
                for rowindex := 0 to upperindex do
                    WriteLn(tempfile,OutputWeightBest[rowindex,colindex]);
                end;
            end;
        else
        begin
            WriteLn(tempfile,RecSetParam.GetNumHidden);

            if RecSetParam.GetNumHidden > 0 then
            begin
                // save the weights for the hidden layer neurons
                upperindex := RecSetParam.GetNumHidden;
                for colindex := 0 to upperindex - 1 do
                    for rowindex := 0 to 128 do
                        WriteLn(tempfile,HiddenWeight[rowindex,colindex]);
                    end;

                // save the weights for the output layer neurons
                for colindex := 0 to RecLoadDBase.GetNumOutputs - 1 do
                    for rowindex := 0 to upperindex do
                        WriteLn(tempfile,OutputWeight[rowindex,colindex]);
                    end;
                end;
            end;
            CloseFile(tempfile);
        end;

        (* load the network weights from a text file where the filename depends on the
        test number *)
        procedure TRecMain.LoadnetworkweightsClick(Sender: TObject);
        var tempfile : textfile;
            filename : string;
            rowindex,colindex,upperindex,tempoutputs,temphidden : integer;
        begin
            upperindex := 128;
            filename := 'c:\TestData\weight'+inttostr(testnumber)+'\txt';
            AssignFile(tempfile,filename);
            Reset(tempfile);
            // read in the number of outputs
            ReadLn(tempfile,tempoutputs);
            // read in the number of hidden layer neurons
            ReadLn(tempfile,temphidden);
            if temphidden > 0 then
            begin
                upperindex := temphidden;
                // load in the weights for the hidden layer neurons
                for colindex := 0 to temphidden - 1 do
                    for rowindex := 0 to 128 do
                        ReadLn(tempfile,HiddenWeight[rowindex,colindex]);
                    end;

                // load in the weights for the output layer neurons
                for colindex := 0 to tempoutputs - 1 do
                    for rowindex := 0 to upperindex do
                        ReadLn(tempfile,OutputWeight[rowindex,colindex]);
                    end;
                end;
            end;
        end;
    end;
end;

```

```

CloseFile(tempfile);
Starttraining1.Enabled := true;
Validatedatabase1.Enabled := true;
Validatecurrentsamples1.Enabled := true;
end;

end.

```

PROGRAM LISTING OF LOADUNIT.PAS

(* This unit loads the data from the specified databases into the matrices which will be used for network training. This unit also performs pre-processing on the loaded data - standardisation is performed with the option of calibrating the system according to test cases where the rig is filled only with water *)

unit loadunit;

interface

uses

Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs, ExtCtrls, StdCtrls, Spin, ComCtrls, Buttons, DBTables, Math, Registry;

type

```

TRecLoadDBase = class(TForm)
    Panel1: TPanel;
    Panel2: TPanel;
    Panel3: TPanel;
    RecTrainStart: TSpinEdit;
    RecTrainStop: TSpinEdit;
    RecTrainName: TComboBox;
    Bevel1: TBevel;
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    Label4: TLabel;
    Bevel2: TBevel;
    Label5: TLabel;
    Label6: TLabel;
    Label7: TLabel;
    Label8: TLabel;
    RecTestName: TComboBox;
    RecTestStart: TSpinEdit;
    RecTestStop: TSpinEdit;
    RecLoadData: TSpeedButton;
    Label9: TLabel;
    Label10: TLabel;
    RecPreProcess: TSpeedButton;
    RecLoadReturn: TSpeedButton;
    RecLoadProgress: TPanel;
    Label11: TLabel;
    RecLoadProgressBar: TProgressBar;
    Session: TSession;
    RecDesiredRecon: TRadioGroup;
    RecLoadStatus: TLabel;
    RecLoadCalibrate: TCheckBox;
    Label12: TLabel;
    Label13: TLabel;
    RecTrainVers: TSpinEdit;
    RecTestVers: TSpinEdit;
    RecDesirePhase: TRadioGroup;
    procedure RecLoadReturnClick(Sender: TObject);
    procedure FormShow(Sender: TObject);
    procedure RecLoadDataClick(Sender: TObject);
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
    procedure RecPreProcessClick(Sender: TObject);
private
    { Private declarations }
    trainnumcases : integer;      // total number of training data samples
    testnumcases : integer;       // total number of testing data samples
    traintotal : integer;         // total training samples loaded
    testtotal : integer;          // total testing samples loaded
    trainnumvrs : integer;        // number of versions for each training
    testnumvrs : integer;         // number of versions for each testing
    // sample
    // sample
    containsgravel : Boolean;      // loaded data contains gravel
end;

```

```

containsair : Boolean;      // loaded data contains air
threephase : Boolean;      // performing a three-phase reconstruction
tomosystem : integer;      // specify which system loaded data is from
opened : Boolean;          // opened databases
numoutputs : integer;      // number of network outputs
TrainInpTable : TTable;    // table to load training input data from
                           // training database
TrainOutTable : TTable;    // table to load training output data from
                           // training database
TestInpTable : TTable;     // table to load testing input data from
                           // testing database
TestOutTable : TTable;     // table to load testing output data from
                           // testing database
trainstartvalues : array[1..5,1..128] of real;
trainstopvalues : array[1..5,1..128] of real;
teststartvalues : array[1..5,1..128] of real;
teststopvalues : array[1..5,1..128] of real;
                           // records of voltages for rig filled only
                           // with water - this will be used to
                           // calibrate the system

procedure PerformCalibrate; // offset the data according to the
                           // calibration data so that only the
                           // changes resulting from inserting a
                           // bubble are stored
procedure ReOrganiseData;  // in the current implementation, all the
                           // data is loaded - however if only want
                           // to train for air-water reconstruction,
                           // then the other data is removed using
                           // this procedure
procedure StandardiseData; // standardise loaded data
function StdDevColumn(colnum:integer; mean:double):double;
                           // find the standard deviation for a column
function MeanColumn(colnum:integer):double;
                           // find the mean of a column

public
{ Public declarations }
function GetTomoSystem: integer; // get which system using
function GetTotalTrain: integer; // get total training points
function GetTotalTest: integer; // get total test points
function GetNumOutputs: integer; // get number of outputs
procedure GetPhases(var contain: Boolean; var contgrav: Boolean);
procedure SetPhases(contain: Boolean; contgrav: Boolean);
                           // get and set phases
end;

var
RecLoadDBase: TRecLoadDBase;
meanrec : array[1..128] of real; // store the means of each column so that
                           // current samples can be pre-processed
                           // according to the same rule used for
                           // the training data
stddevrec : array[1..128] of real; // store the standard deviations of each
                           // column
custload : Boolean;
trainstartindex : array[1..5] of integer;
trainstopindex : array[1..5] of integer;
teststartindex : array[1..5] of integer;
teststopindex : array[1..5] of integer;
numtrainregions : integer;
numtestregions : integer;

implementation

uses mainunit, calibset;

{$R *.DFM}

(* ----- DATA LOADING PROCEDURES ----- *)
(* this procedure loads the data in the specified databases into the matrices that
will be used for network training and network testing. To access the data,
temporary tables are created for the databases and opened.
NOTE: the data is loaded as follows
FOR THREE PHASE IMAGE RECONSTRUCTION
    O0 water O1 gravel O2 air
FOR VOLUME PREDICTION
    O0 air O1 gravel
FOR VOLUME DATA
    O0 air O1 gravel O2 water
FOR TWO PHASE IMAGE RECONSTRUCTION
    -1 water target +1 other phase

```

```

Specify the table sizes as if all the data were to be loaded. Then, depending
on whether a test case contains air, gravel or mixture load the data into the
table. This will allow the standard calibration feature to correctly calculate
the offset for two-phase networks where the data is not continuous.
*)
procedure TRecLoadDBase.RecLoadDataClick(Sender: TObject);
var row, col, tempvalue, imgcol: integer;
    s1,s2 : string;
    trainversions,trainnumber,testnumber,testversions,traincalib,testcalib:integer;
    index : real;
    tempair, tempgravel : Boolean;
begin
    // if wish to perform calibration then show calibration setup form
    // so that the user can specify which training data points represent
    // the rig being filled only with water
    if RecLoadCalibrate.Checked then
        CalibSetup.ShowModal;

    if CalibSetup.ModalResult <> mrCancel then
        begin
            Update;
            // create the tables for the databases so that the data can be accessed
            TrainInpTable := TTable.Create(Self);
            TrainOutTable := TTable.Create(Self);
            TestInpTable := TTable.Create(Self);
            TestOutTable := TTable.Create(Self);
            // set the database names to those specified by the user
            TrainInpTable.DatabaseName := RecTrainName.Text;
            TrainOutTable.DatabaseName := RecTrainName.Text;
            TestInpTable.DatabaseName := RecTestName.Text;
            TestOutTable.DatabaseName := RecTestName.Text;
            TrainInpTable.TableName := 'InpTrain.db';
            TrainOutTable.TableName := 'OutTrain.db';
            TestInpTable.TableName := 'InpTrain.db';
            TestOutTable.TableName := 'OutTrain.db';
            // open the tables
            TrainInpTable.Open;
            TrainOutTable.Open;
            TestInpTable.Open;
            TestOutTable.Open;

            InputData := nil;
            OutputData := nil;
            VolumeData := nil;
            // calculate the number of training and testing points to load
            traintotal := (RecTrainStop.Value - RecTrainStart.Value + 1) *
                RecTrainVers.Value;
            testtotal := (RecTestStop.Value - RecTestStart.Value + 1) *
                RecTestVers.Value;
            // initialise matrices into which data will be loaded
            SetLength(InputData,traintotal+testtotal,129);
            SetLength(VolumeData,traintotal+testtotal,3);

            threephase := false;
            containsair := false;
            containsgravel := false;
            s2 := '';

            for row := 1 to 5 do
                begin
                    for col := 1 to 128 do
                        begin
                            trainstartvalues[row,col] := 0;
                            trainstopvalues[row,col] := 0;
                            teststartvalues[row,col] := 0;
                            teststopvalues[row,col] := 0;
                        end;
                    end;
                end;

            // calculate the maximum number of training and testing versions
            TrainInpTable.Last;
            trainnumcases := integer(TrainInpTable.FieldName('TestNo').Value);
            trainnumvrs := 1000 * (TrainInpTable.FieldName('TestNo').Value -
                trainnumcases);
            TestInpTable.Last;
            testnumcases := integer(TestInpTable.FieldName('TestNo').Value);
            testnumvrs := 1000 * (TestInpTable.FieldName('TestNo').Value -
                testnumcases);

            // load the water-only endpoints to provide calibration

```

```

// for the start of the training data
for traincalib := 1 to numtrainregions do
begin
  for trainversions := 1 to trainnumvrs do
  begin
    index := trainstartindex[traincalib] + trainversions/1000;
    TrainInpTable.FindKey([index]);
    for col := 1 to 128 do
    begin
      s1 := 'Voltage'+inttostr(col);
      trainstartvalues[traincalib,col] :=
        trainstartvalues[traincalib,col]+
        TrainInpTable.FieldByName(s1).Value;
    end;
  end;
end;

// load the water-only endpoints to provide calibration
// for the end of the training data
for traincalib := 1 to numtrainregions do
begin
  for trainversions := 1 to trainnumvrs do
  begin
    index := trainstopindex[traincalib] + trainversions/1000;
    TrainInpTable.FindKey([index]);
    for col := 1 to 128 do
    begin
      s1 := 'Voltage'+inttostr(col);
      trainstopvalues[traincalib,col] :=
        trainstopvalues[traincalib,col]+
        TrainInpTable.FieldByName(s1).Value;
    end;
  end;
end;

// load the water-only testpoints to provide a calibration
// for the start of the test data
for testcalib := 1 to numtestregions do
begin
  for testversions := 1 to testnumvrs do
  begin
    index := teststartindex[testcalib] + testversions/1000;
    TestInpTable.FindKey([index]);
    for col := 1 to 128 do
    begin
      s1 := 'Voltage'+inttostr(col);
      teststartvalues[testcalib,col] :=
        teststartvalues[testcalib,col]+
        TestInpTable.FieldByName(s1).Value;
    end;
  end;
end;

// load the water-only testpoints to provide a calibration
// for the end of the test data
for testcalib := 1 to numtestregions do
begin
  for testversions := 1 to testnumvrs do
  begin
    index := teststopindex[testcalib] + testversions/1000;
    TestInpTable.FindKey([index]);
    for col := 1 to 128 do
    begin
      s1 := 'Voltage'+inttostr(col);
      teststopvalues[testcalib,col] :=
        teststopvalues[testcalib,col]+
        TestInpTable.FieldByName(s1).Value;
    end;
  end;
end;

for row := 1 to 5 do
  for col := 1 to 128 do
  begin
    trainstartvalues[row,col] := trainstartvalues[row,col]/trainnumvrs;
    trainstopvalues[row,col] := trainstopvalues[row,col]/trainnumvrs;
    teststartvalues[row,col] := teststartvalues[row,col]/testnumvrs;
    teststopvalues[row,col] := teststopvalues[row,col]/testnumvrs;
  end;
end;

// set the sizes of the matrices according to the desired output

```

```

if RecDesiredRecon.ItemIndex = 0 then
begin
  // three-phase measurement requires 1-of-C encoding for
  // each output pixel therefore 264 outputs
  if RecDesirePhase.ItemIndex = 0 then
  begin
    SetLength(OutputData,traintotal+testtotal,264);
    SetLength(OutputNetwork,traintotal+testtotal,264);
    numoutputs := 264;
    threephase := true;
    containsair := true;
    containsgravel := true;
    s2 := ' for a three-phase measurement';
  end
  // two-phase image reconstruction only requires a single
  // output per pixel therefore 88 outputs
  else
  begin
    SetLength(OutputData,traintotal+testtotal,88);
    SetLength(OutputNetwork,traintotal+testtotal,88);
    numoutputs := 88;
    if RecDesirePhase.ItemIndex = 1 then
    begin
      containsair := true;
      s2 := ' for an air-water two-phase measurement';
    end
    else
    begin
      containsgravel := true;
      s2 := ' for a gravel-water two-phase measurement';
    end;
  end;
end;

// volume fraction reconstruction requires two outputs - one for
// air and one for gravel
else
begin
  SetLength(OutputData,traintotal+testtotal,2);
  SetLength(OutputNetwork,traintotal+testtotal,88);
  numoutputs := 2;
end;

RecLoadProgress.Visible := true;
RecLoadProgress.Update;
RecLoadProgressBar.Max := traintotal + testtotal;
RecLoadProgressBar.Position := 0;

row := 0;
// load the training data
// for the two-phase predictions, load the input data but set
// the output to zero for incorrect phase inputs to serve as a
// flag so that this data can be overwritten at a later stage
for trainversions := 1 to RecTrainVers.Value do
begin
  for trainnumber := RecTrainStart.Value to RecTrainStop.Value do
  begin
    tempair := false;
    tempgravel := false;
    index := trainnumber + trainversions/1000;
    TrainOutTable.FindKey([index]);
    TrainInpTable.FindKey([index]);
    TomoSystem := TrainInpTable.FieldByName('System').Value;
    InputData[row,0] := 1;
    // load the input data
    for col := 1 to 128 do
    begin
      s1 := 'Voltage'+inttostr(col);
      InputData[row,col] :=
        TrainInpTable.FieldByName(s1).Value;
    end;

    if RecDesiredRecon.ItemIndex = 0 then
    begin
      // if desired output is image reconstruction
      // calculate what phases are in this training case
      for col := 1 to 100 do
      begin
        s1 := 'Pixel'+inttostr(col);
        tempvalue := TrainOutTable.FieldByName(s1).Value;
        if tempvalue = 0 then
          tempair := true

```



```

else
begin
    if tempvalue = 1 then
        OutputData[row,imgcol] := -1
    else
        OutputData[row,imgcol] := 1;
        inc(imgcol);
    end;
end;
end;
// else set a flag so that the data can be removed
// at a later stage
else if ((RecDesirePhase.ItemIndex = 1)and(tempgravel))or
((RecDesirePhase.ItemIndex = 2)and(tempair))then
begin
    OutputData[row,imgcol] := 0;
    inc(imgcol);
    if imgcol = 88 then
        imgcol := 0;
    end;
end;
end;
// else desired output is volume fraction prediction
else
begin
    // determine the phases in the current test case
    if TestOutTable.FieldByName('AirVolume').Value > 0 then
        tempair := true;
    if TestOutTable.FieldByName('GravelVolume').Value > 0 then
        tempgravel := true;

    // if the phases are the correct phases for the desired
    // reconstruction then load the data normally
    if ((RecDesirePhase.ItemIndex = 1)and(tempair)
and(not tempgravel))or((RecDesirePhase.ItemIndex = 2)
and(not tempair)and(tempgravel))or
((not tempair)and(not tempgravel))or
(RecDesirePhase.ItemIndex = 0)then
begin
    OutputData[row,0] :=
        TestOutTable.FieldByName('AirVolume').Value;
    OutputData[row,1] :=
        TestOutTable.FieldByName('GravelVolume').Value;
end
    // else set a flag so that the test case can be removed
    // at a later stage
    else if ((RecDesirePhase.ItemIndex = 1)and(tempgravel))or
    ((RecDesirePhase.ItemIndex = 2)and(tempair))then
begin
        OutputData[row,0] := -1;
        OutputData[row,1] := -1;
    end;
end;

VolumeData[row,0] := TestOutTable.FieldByName('AirVolume').Value;
VolumeData[row,1] := TestOutTable.FieldByName('GravelVolume').Value;
VolumeData[row,2] := TestOutTable.FieldByName('WaterVolume').Value;
inc(row);
RecLoadProgressBar.StepIt;
Application.ProcessMessages;
end;
end;
RecLoadProgress.Visible := false;
opened := true;

// display a summary of the loading results
if TomoSystem = 0 then
    s1 := 'upper'
else
    s1 := 'lower';

RecLoadStatus.Caption := 'Loaded '+inttostr(traintotal+testtotal)+
    ' datapoints from the '+s1+' system'+s2;
RecLoadStatus.Visible := true;

RecMain.Viewdatabase1.Enabled := true;
RecMain.Writedataatextfile1.Enabled := true;
end;
end;

(* ----- DATA ACCESS PROCEDURES ----- *)
function TRecLoadDBase.GetTotalTrain: integer;
begin
    GetTotalTrain := traintotal;
end;

function TRecLoadDBase.GetTotalTest: integer;
begin
    GetTotalTest := testtotal;
end;

procedure TRecLoadDBase.GetPhases(var contain, contgrav: Boolean);
begin
    contain := containsair;
    contgrav := containsgravel;
end;

function TRecLoadDBase.GetNumOutputs: integer;
begin
    GetNumOutputs := numoutputs;
end;

function TRecLoadDBase.GetTomoSystem: integer;
begin
    GetTomoSystem := tomosystem;
end;

procedure TRecLoadDBase.SetPhases(contain, contgrav: Boolean);
begin
    containsair := contain;
    containsgravel := contgrav;
end;

(* ----- PRE-PROCESSING PROCEDURES ----- *)
(* pre-process the loaded data by optionally performing calibration and then
standardise *)
procedure TRecLoadDBase.RecPreProcessClick(Sender: TObject);
begin
    RecLoadProgressBar.Position := 0;
    RecLoadProgressBar.Max := traintotal + testtotal;
    RecLoadProgress.Visible := true;
    RecLoadProgress.Update;

    if RecLoadCalibrate.Checked then
        PerformCalibrate;

    // if only desire two-phase reconstruction, then need to remove those
    // test cases relating to the other phase as well as three-phase
    // mixtures
    if RecDesirePhase.ItemIndex <> 0 then
        ReOrganiseData;

    RecLoadProgressBar.Position := 0;
    RecLoadProgressBar.Max := 128;
    StandardiseData;
    RecLoadProgress.Visible := false;
end;

(* standardise the data by subtracting the mean of each column from the readings
in that column and dividing by the standard deviation for that column *)
procedure TRecLoadDBase.StandardiseData;
var rowindex, colindex: integer;
    mean, stddev: double;
begin
    for colindex := 1 to 128 do
        begin
            // find the mean of the column and keep a record for future tests
            mean := MeanColumn(colindex);
            meanrec[colindex] := mean;
            // find the standard deviation of the column and keep a record for
            // future tests
            stddev := StdDevColumn(colindex,mean);
            stddevrec[colindex] := stddev;
            // pre-process the data by subtracting the mean and dividing by the
            // standard deviation
            for rowindex := 0 to (traintotal+testtotal-1) do
                InputData[rowindex,colindex] := (InputData[rowindex,colindex] -
                    mean)/stddev;
            RecLoadProgressBar.StepIt;
        end;
    end;
end;

```

```

end;

(* calculate the standard deviation of a column of training data *)
function TRecLoadDBase.StdDevColumn(colnum:integer; mean:double):double;
var i : integer;
    sum: double;
begin
    sum := 0;
    for i:= 0 to (traintotal - 1) do
        sum := sum + IntPower(InputData[i,colnum] - mean,2);
    end;

    StdDevColumn := sqrt(sum/(traintotal-1));
end;

(* calculate the mean of a column of training data *)
function TRecLoadDBase.MeanColumn(colnum:integer):double;
var sum:double;
    i : integer;
begin
    sum := 0;
    for i := 0 to (traintotal - 1) do
        sum := sum + InputData[i,colnum];
    end;
    MeanColumn := sum/traintotal;
end;

(* calibration works as follows - before start generating a training database,
a set of frames is taken for the rig filled only with water. During database
generation, drift takes place in the readings as a result of the properties of
the water changing, temperature changes, etc (NOTE: training database generation
typically takes place over a couple of days). At the end of training database
generation, another set of frames is captured for the rig filled only with water.
These provide the two endpoints and the calibration for points in-between are
calculated as a linear interpolation between these two points. The process is
then repeated for the testing database *)
procedure TRecLoadDBase.PerformCalibrate;
var rowindex,colindex,region,numrepeat : integer;
    driftcomp : real;
begin
    rowindex := 0;
    region := 1;
    numrepeat := 0;
    while (rowindex < traintotal) do
        begin
            for colindex := 1 to 128 do
                begin
                    driftcomp := (trainstopvalues[region,colindex] -
trainstartvalues[region,colindex])*(rowindex-
(trainstartindex[region]+numrepeat*RecTrainStop.Value)+1)
/(trainstopindex[region]-trainstartindex[region]+1)
+ trainstartvalues[region,colindex];
                    InputData[rowindex,colindex] := InputData[rowindex,colindex]
-driftcomp;
                end;
            end;
            inc(rowindex);
            RecLoadProgressBar.StepIt;
            if rowindex=(trainstopindex[region]+(numrepeat*RecTrainStop.Value))then
                begin
                    inc(region);
                    if region > numtrainregions then
                        begin
                            region := 1;
                            inc(numrepeat);
                        end;
                end;
            end;
        end;

        region := 1;
        numrepeat := 0;

        while (rowindex < traintotal+testtotal) do
            begin
                for colindex := 1 to 128 do
                    begin
                        driftcomp := (teststopvalues[region,colindex] -
teststartvalues[region,colindex])*(rowindex-
(teststartindex[region]+numrepeat*RecTestStop.Value+traintotal)+1)
/(teststopindex[region]-teststartindex[region]+1)
+ teststartvalues[region,colindex];
                        InputData[rowindex,colindex] := InputData[rowindex,colindex]
-driftcomp;
                    end;
                end;
            end;
        end;
    end;

```

```

inc(rowindex);
RecLoadProgressBar.StepIt;
if rowindex=(teststopindex[region]+(numrepeat*RecTestStop.Value)
+traintotal)then
    begin
        inc(region);
        if region > numtestregions then
            begin
                region := 1;
                inc(numrepeat);
            end;
        end;
    end;
end;

end;

(* not all the data that was loaded can be used for a two-phase measurement
therefore this routine effectively copies over the data that was flagged as
incorrect phase during the data loading procedure *)
procedure TRecLoadDBase.ReOrganiseData;
var origindex, newindex, colindex,temptraintotal,temptesttotal : integer;
begin
    newindex := 0;
    temptraintotal := 0;
    temptesttotal := 0;
    RecLoadProgressBar.Max := traintotal + testtotal;
    RecLoadProgressBar.Position := 0;
    for origindex := 0 to traintotal+testtotal-1 do
        begin
            // if image reconstruction and valid test case then copy the data
            // across and increment newindex
            // if volume fraction reconstruction and valid test case then copy
            // the data across and increment newindex
            if ((OutputData[origindex,0]<>0)and(RecDesiredRecon.ItemIndex = 0))or
((OutputData[origindex,0]<->1)and(RecDesiredRecon.ItemIndex = 1))then
                begin
                    for colindex := 0 to 128 do
                        InputData[newindex,colindex] := InputData[origindex,colindex];
                    end;
                    for colindex := 0 to numoutputs-1 do
                        OutputData[newindex,colindex] := OutputData[origindex,colindex];
                    end;
                    for colindex := 0 to 2 do
                        VolumeData[newindex,colindex] := VolumeData[origindex,colindex];
                    end;
                    inc(newindex);
                    if origindex < traintotal then
                        inc(temptraintotal)
                    else
                        inc(temptesttotal);
                    end;
                end;
            // else simply increment origindex
            RecLoadProgressBar.StepIt;
        end;
        InputData := Copy(InputData,0,newindex);
        OutputData := Copy(OutputData,0,newindex);
        OutputNetwork := Copy(OutputNetwork,0,newindex);
        VolumeData := Copy(VolumeData,0,newindex);
        traintotal := temptraintotal;
        testtotal := temptesttotal;
    end;

    (* ----- GENERAL PROCEDURES ----- *)
    (* close the form *)
    procedure TRecLoadDBase.RecLoadReturnClick(Sender: TObject);
    begin
        RecLoadStatus.Visible := false;
        RecMain.Savenetworkpreprocessing1.Enabled := true;
        Close;
    end;

    (* when show form, load all the possible database names into the combo boxes - a
user then chooses which databases contain the training and testing data. The
previous settings used are also loaded from the registry *)
    procedure TRecLoadDBase.FormShow(Sender: TObject);
    var templist: TStringList;
        i : integer;
        Reg : TRegistry;
        KeyGood : Boolean;
    begin
        custload := false;
        templist := TStringList.Create;
        // get a list of all available database names
        Session.GetAliasNames(templist);
    end;

```

```

for i := 0 to templist.Count - 1 do
begin
    RecTrainName.Items.Add(templist[i]);
    RecTestName.Items.Add(templist[i]);
end;
templist.Free;
opened := false;

// load the previously used settings from the registry
Reg := TRegistry.Create;
try
    KeyGood := Reg.OpenKey('Software\Neural', False);
    // if the key exists then read from the key, else it is the
    // first time the program is running so do not attempt
    // to read from the registry
    if KeyGood then
    begin
        RecDesiredRecon.ItemIndex := Reg.ReadInteger('OutputDesired');
        RecTrainName.Text := Reg.ReadString('TrainDatabase');
        RecTestName.Text := Reg.ReadString('TestDatabase');
        RecTrainStart.Value := Reg.ReadInteger('TrainStart');
        RecTrainStop.Value := Reg.ReadInteger('TrainStop');
        RecTestStart.Value := Reg.ReadInteger('TestStart');
        RecTestStop.Value := Reg.ReadInteger('TestStop');
        RecLoadCalibrate.Checked := Reg.ReadBool('Calibrate');
        RecTrainVers.Value := Reg.ReadInteger('TrainVers');
        RecTestVers.Value := Reg.ReadInteger('TestVers');
        RecDesirePhase.ItemIndex := Reg.ReadInteger('DesiredLoadPhase');
    end;
finally
    Reg.Free;
end;
end;

(* when close form, write the current parameters to the registry. If the
data has been loaded, then free the resources allocated to the temporary tables *)
procedure TRecLoadDBase.FormClose(Sender: TObject;
var Action: TCloseAction);
var Reg: TRegistry;
begin
    Reg := TRegistry.Create;
    try
        Reg.OpenKey('Software\Neural', True);
        Reg.WriteInteger('OutputDesired', RecDesiredRecon.ItemIndex);
        Reg.WriteString('TrainDatabase', RecTrainName.Text);
        Reg.WriteString('TestDatabase', RecTestName.Text);
        Reg.WriteInteger('TrainStart', RecTrainStart.Value);
        Reg.WriteInteger('TrainStop', RecTrainStop.Value);
        Reg.WriteInteger('TestStart', RecTestStart.Value);
        Reg.WriteInteger('TestStop', RecTestStop.Value);
        Reg.WriteInteger('TrainVers', RecTrainVers.Value);
        Reg.WriteInteger('TestVers', RecTestVers.Value);
        Reg.WriteInteger('DesiredLoadPhase', RecDesirePhase.ItemIndex);
        Reg.WriteBool('Calibrate', RecLoadCalibrate.Checked);
    finally
        Reg.Free;
    end;

    // if the data has been loaded then close the tables and free the
    // allocated resources
    if opened then
    begin
        TrainInpTable.Close;
        TrainOutTable.Close;
        TestInpTable.Close;
        TestOutTable.Close;
        TrainInpTable.Free;
        TrainOutTable.Free;
        TestInpTable.Free;
        TestOutTable.Free;
    end;
end;
end.

```

PROGRAM LISTING OF CALIBSET.PAS

(* This unit simply provides an interface for the user to stipulate the

endpoints for the calibration feature - these endpoints represent test cases where the rig was filled only with water and provide a means of compensating for the drift that takes place over the days of generating training and test databases. *)

unit calibset;

interface

uses

Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs, StdCtrls, ExtCtrls;

type

```

TCalibSetup = class(TForm)
    Panel1: TPanel;
    CalibOK: TButton;
    CalibCancel: TButton;
    MainPanel: TPanel;
    MainPrompt: TLabel;
    MainEdit: TEdit;
    InputPanel: TPanel;
    InputPrompt: TLabel;
    Label1: TLabel;
    Label2: TLabel;
    InputStart: TEdit;
    InputStop: TEdit;
    procedure FormShow(Sender: TObject);
    procedure CalibOKClick(Sender: TObject);
private
    { Private declarations }
    currentindex: integer;
    currentstage: integer;
public
    { Public declarations }
end;

```

var

CalibSetup: TCalibSetup;

implementation

uses loadunit;

(\$R *.DFM)

(* ----- GENERAL PROCEDURES ----- *)

(* initialise variables *)

procedure TCalibSetup.FormShow(Sender: TObject);

begin

```

    MainPanel.Visible := true;
    InputPanel.Visible := false;
    MainEdit.Text := '1';
    MainPrompt.Caption := 'Specify the number of training regions:';
    currentindex := 1;
    currentstage := 1;
    MainEdit.SetFocus;
    CalibOK.ModalResult := mrNone;
end;

```

end;

(* collect information regarding the calibration points for both the training and testing databases *)

procedure TCalibSetup.CalibOKClick(Sender: TObject);

begin

```

    if currentstage = 1 then
    begin
        numtrainregions := strtoint(MainEdit.Text);
        currentstage := 2;
        InputPanel.Visible := true;
        InputPrompt.Caption := 'Specify the bounds for region '+
            inttostr(currentindex)+':';
        InputStart.Text := '1';
        InputStart.Enabled := false;
        InputStop.Text := '1';
        InputStop.SetFocus;
    end
end

```

```

    else if currentstage = 2 then
    begin

```

```

        trainstartindex[currentindex] := strtoint(InputStart.Text);
        trainstopindex[currentindex] := strtoint(InputStop.Text);
        inc(currentindex);
    end
end;

```

```

if currentIndex > numtrainregions then
begin
    InputPanel.Visible := false;
    MainPrompt.Caption :=
        'Specify the number of testing regions: ';
    currentIndex := 1;
    MainEdit.Text := '1';
    currentstage := 3;
    MainEdit.SetFocus;
end
else
begin
    InputPrompt.Caption := 'Specify the bounds for region '+'
        intostr(currentindex)+' ';
    InputStart.Text := intostr(strtoint(InputStop.Text)+1);
    InputStart.Enabled := false;
    InputStop.Text := '';
    InputStop.SetFocus;
end;
end
else if currentstage = 3 then
begin
    numtestregions := strtoint(MainEdit.Text);
    currentstage := 4;
    InputPanel.Visible := true;
    InputPrompt.Caption := 'Specify the bounds for region '+'
        intostr(currentindex)+' ';
    InputStart.Text := '1';
    InputStart.Enabled := false;
    InputStop.Text := '1';
    InputStop.SetFocus;
end
else if currentstage = 4 then
begin
    teststartindex[currentindex] := strtoint(InputStart.Text);
    teststopindex[currentindex] := strtoint(InputStop.Text);
    inc(currentindex);
    if currentindex > numtestregions then
    begin
        CalibOK.ModalResult := mrOK;
        currentstage := 5;
        CalibOK.Click;
    end
    else
    begin
        InputPrompt.Caption := 'Specify the bounds for region '+'
            intostr(currentindex)+' ';
        InputStart.Text := intostr(strtoint(InputStop.Text)+1);
        InputStart.Enabled := false;
        InputStop.Text := '';
        InputStop.SetFocus;
    end;
end;
end;
end.

```

PROGRAM LISTING OF VIEWUNIT.PAS

(* This unit displays the loaded data, both the desired network output and the corresponding sampled voltages *)

unit viewunit;

interface

uses

Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs, Buttons, ExtCtrls, TeEngine, Series, TeeProcs, Chart, StdCtrls, Grids, DBGrids, Db, Tomo;

type

TResultTable = (capacitance, resistance);
TRecViewMain = class(TForm)
 Panel1: TPanel;
 RecViewCapTable: TStringGrid;
 Label3: TLabel;
 Label5: TLabel;

Label4: TLabel;
RecViewResTable: TStringGrid;
RecViewNext: TSpeedButton;
RecViewPrevious: TSpeedButton;
RecViewVolLabel: TLabel;
RecViewVolume: TChart;
Series1: TBarSeries;
Series2: TBarSeries;
Series3: TBarSeries;
RecViewReturn: TSpeedButton;
RecViewInputTomo: TTomo;
RecViewImgLabel: TLabel;
ScrollTimer: TTimer;
UseContinous: TCheckBox;
procedure RecViewReturnClick(Sender: TObject);
procedure FormCreate(Sender: TObject);
procedure FormShow(Sender: TObject);
procedure RecViewNextClick(Sender: TObject);
procedure RecViewPreviousClick(Sender: TObject);
procedure ScrollTimerTimer(Sender: TObject);
private
 { Private declarations }
 currentIndex : integer; // current index into database
 procedure DisplayData; // display the desired network output
 procedure UpdateTables; // update the tables to show the
 // corresponding input voltages
public
 { Public declarations }
end;

var
 RecViewMain: TRecViewMain;

implementation

uses loadunit, mainunit;

(\$R *.DFM)

(* ----- DATABASE VIEWING PROCEDURES ----- *)
(* move to the next test case in the database by incrementing currentIndex and calling DisplayData - if at the last element in the database then disable the button *)
procedure TRecViewMain.RecViewNextClick(Sender: TObject);
begin

inc(currentindex);
 RecViewPrevious.Enabled := true;
 if currentIndex =
 (RecLoadDBase.GetTotalTest + RecLoadDBase.GetTotalTrain - 1) then
 begin
 RecViewNext.Enabled := false;
 UseContinous.Checked := false;

end;
 DisplayData;
end;

(* move to the previous test case in the database by decrementing currentIndex and calling DisplayData - if first element in the database then disable the button *)

procedure TRecViewMain.RecViewPreviousClick(Sender: TObject);
begin

dec(currentindex);
 if currentIndex = 0 then
 begin
 UseContinous.Checked := false;
 RecViewPrevious.Enabled := False;

end;
 RecViewNext.Enabled := true;

DisplayData;

end;

(* display the desired network output for the current test case in the database *)

procedure TRecViewMain.DisplayData;

var contain, contgrav : Boolean;

index, row, col, top, bottom : integer;

tempout : TOutput;

begin
 RecViewVolLabel.Caption := 'Volume fractions for test '
 + intostr(currentindex+1)+'';
 if RecLoadDBase.RecDesiredRecon.ItemIndex = 0 then
 begin


```

// for the image reconstruction, use a TTomato component to display
// the desired network output
RecViewImgLabel.Caption := 'Desired output for test '
+inttostr(currentindex+1)+'.';

RecLoadDBase.GetPhases(contair,contgrav);
RecViewInputTomo.ResetDesired;

for col := 0 to 9 do
begin
  for row := 0 to 9 do
  begin
    tempout[col,row] := outpipe;
  end;
end;

index := 0;
for col := 0 to 9 do
begin
  case col of
    0: begin top := 2; bottom := 7; end;
    1: begin top := 1; bottom := 8; end;
    2..7: begin top := 0; bottom := 9; end;
    8: begin top := 1; bottom := 8; end;
    9: begin top := 2; bottom := 7; end;
  end;

  for row := top to bottom do
  begin
    if contair and contgrav then
    begin
      // if three-phase then decode 1-of-C encoding
      // employed
      if OutputData[currentindex,index] = 1 then
        tempout[col,row] := water
      else if OutputData[currentindex,index+1] = 1 then
        tempout[col,row] := gravel
      else
        tempout[col,row] := air;
      inc(index,3);
    end
    else
    begin
      // else simply two-phase
      if OutputData[currentindex,index] = 1 then
      begin
        if contair then
          tempout[col,row] := air
        else
          tempout[col,row] := gravel;
        end
      else if OutputData[currentindex,index] = -1 then
        tempout[col,row] := water
      else tempout[col,row] := outpipe;
      inc(index)
    end;
  end;
end;

// set the TTomato component and force a repaint
RecViewInputTomo.Desired := tempout;
RecViewInputTomo.Repaint;
// display the volume fraction data using the VolumeData matrix
RecViewVolume.SeriesList[0].YValue[0] := VolumeData[currentindex,2];
RecViewVolume.SeriesList[1].YValue[0] := VolumeData[currentindex,0];
RecViewVolume.SeriesList[2].YValue[0] := VolumeData[currentindex,1];

end
else
begin
  // for volume fraction prediction, use a chart to display the
  // desired volume fractions
  if OutputData[currentindex,0] <> -1 then
  begin
    RecViewVolume.SeriesList[0].YValue[0] :=
      (100-(OutputData[currentindex,0]+OutputData[currentindex,1]));
    RecViewVolume.SeriesList[1].YValue[0] := OutputData[currentindex,0];
    RecViewVolume.SeriesList[2].YValue[0] := OutputData[currentindex,1];
  end
  else
    begin
      RecViewVolume.SeriesList[0].YValue[0] := 0;
      RecViewVolume.SeriesList[1].YValue[0] := 0;
      RecViewVolume.SeriesList[2].YValue[0] := 0;
    end;
  end;
end;

end;
UpdateTables;

end;

(* display the input voltages in a tabular format *)
procedure TRecViewMain.UpdateTables;
var row, col, index, overall : integer;
    result : TResultTable;
    s1,s2 : string;
begin
  index := 1;
  for overall := 1 to 16 do
  begin
    case overall of
      1 : begin
          row := 4;    result := capacitance;
        end;
      2 : begin
          row := 4;    result := resistance;
        end;
      3 : begin
          row := 3;    result := capacitance;
        end;
      4 : begin
          row := 3;    result := resistance;
        end;
      5 : begin
          row := 2;    result := capacitance;
        end;
      6 : begin
          row := 2;    result := resistance;
        end;
      7 : begin
          row := 1;    result := capacitance;
        end;
      8 : begin
          row := 1;    result := resistance;
        end;
      9 : begin
          row := 5;    result := resistance;
        end;
      10: begin
          row := 5;    result := capacitance;
        end;
      11: begin
          row := 6;    result := resistance;
        end;
      12: begin
          row := 6;    result := capacitance;
        end;
      13: begin
          row := 7;    result := resistance;
        end;
      14: begin
          row := 7;    result := capacitance;
        end;
      15: begin
          row := 8;    result := resistance;
        end;
      16: begin
          row := 8;    result := capacitance;
        end;
    end;
  end;
  for col := 1 to 8 do
  begin
    s1 := 'Voltage'+inttostr(index);
    s2 := Format('%2.5f',[InputData[currentindex,index]]);
    if result = capacitance then
      RecViewCapTable.Cells[col,row] := s2
    else
      RecViewResTable.Cells[col,row] := s2;
    inc(index);
  end;
end;
end;
end;

```

```
(* ----- GENERAL PROCEDURES ----- *)
(* close the form *)
procedure TRecViewMain.RecViewReturnClick(Sender: TObject);
begin
    Close;
end;

(* when create the form, initialise the tables and the chart representing the
volume fractions *)
procedure TRecViewMain.FormCreate(Sender: TObject);
begin
    RecViewResTable.Cells[0,1] := 'TxA';
    RecViewResTable.Cells[0,2] := 'TxB';
    RecViewResTable.Cells[0,3] := 'TxC';
    RecViewResTable.Cells[0,4] := 'TxD';
    RecViewResTable.Cells[0,5] := 'TxE';
    RecViewResTable.Cells[0,6] := 'TxF';
    RecViewResTable.Cells[0,7] := 'TxG';
    RecViewResTable.Cells[0,8] := 'TxH';
    RecViewResTable.Cells[1,0] := 'RxA';
    RecViewResTable.Cells[2,0] := 'RxB';
    RecViewResTable.Cells[3,0] := 'RxC';
    RecViewResTable.Cells[4,0] := 'RxD';
    RecViewResTable.Cells[5,0] := 'RxE';
    RecViewResTable.Cells[6,0] := 'RxF';
    RecViewResTable.Cells[7,0] := 'RxG';
    RecViewResTable.Cells[8,0] := 'RxH';
    RecViewCapTable.Cells[0,1] := 'TxA';
    RecViewCapTable.Cells[0,2] := 'TxB';
    RecViewCapTable.Cells[0,3] := 'TxC';
    RecViewCapTable.Cells[0,4] := 'TxD';
    RecViewCapTable.Cells[0,5] := 'TxE';
    RecViewCapTable.Cells[0,6] := 'TxF';
    RecViewCapTable.Cells[0,7] := 'TxG';
    RecViewCapTable.Cells[0,8] := 'TxH';
    RecViewCapTable.Cells[1,0] := 'RxA';
    RecViewCapTable.Cells[2,0] := 'RxB';
    RecViewCapTable.Cells[3,0] := 'RxC';
    RecViewCapTable.Cells[4,0] := 'RxD';
    RecViewCapTable.Cells[5,0] := 'RxE';
    RecViewCapTable.Cells[6,0] := 'RxF';
    RecViewCapTable.Cells[7,0] := 'RxG';
    RecViewCapTable.Cells[8,0] := 'RxH';
    RecViewVolume.SeriesList[0].Add(100);
    RecViewVolume.SeriesList[1].Add(0);
    RecViewVolume.SeriesList[2].Add(0);
end;

(* when show form, display either an image of the vessel cross-section or a
chart of the volume fractions - depending on what the desired network output
is *)
procedure TRecViewMain.FormShow(Sender: TObject);
begin
    currentIndex := 0;
    RecViewPrevious.Enabled := false;
    RecViewNext.Enabled := true;
    if RecLoadDBase.RecDesiredRecon.ItemIndex = 0 then
    begin
        RecViewImgLabel.Visible := true;
        RecViewInputTomo.Visible := true;
    end
    else
    begin
        RecViewImgLabel.Visible := false;
        RecViewInputTomo.Visible := false;
    end;
    DisplayData;
end;

(* scroll through all the test cases *)
procedure TRecViewMain.ScrollTimerTimer(Sender: TObject);
begin
    if (RecViewNext.Down) and (UseContinous.Checked) then
        RecViewNextClick(sender);
    if (RecViewPrevious.Down) and (UseContinous.Checked) then
        RecViewPreviousClick(sender);
end;
end.
```

PROGRAM LISTING OF PARAMUNIT.PAS

```
(* This unit is used to set the desired network training parameters. When the
form is opened, the previous parameters used are automatically loaded from the
registry *)
unit paramunit;
```

```
interface
```

```
uses
```

```
Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
StdCtrls, Spin, ExtCtrls, Buttons, Registry;
```

```
type
```

```
TRecSetParam = class(TForm)
    Panel1: TPanel;
    RecSetGenParam: TGroupBox;
    RecSetEarlyParam: TGroupBox;
    RecSetNetParam: TGroupBox;
    RecSetModelParam: TGroupBox;
    Label1: TLabel;
    Label3: TLabel;
    RecSetMaxEpochs: TEdit;
    RecSetNumHidden: TSpinEdit;
    RecPerformEarlyStop: TCheckBox;
    RecPerformModel: TCheckBox;
    RecUseGradDescent: TRadioButton;
    RecUseRPROP: TRadioButton;
    Label4: TLabel;
    Label5: TLabel;
    RecSetLearn: TEdit;
    RecSetMomentum: TEdit;
    RecRPropParam: TPanel;
    Label6: TLabel;
    Label7: TLabel;
    RecSetDeltaInit: TEdit;
    RecSetDeltaMax: TEdit;
    Label8: TLabel;
    RecNumFails: TSpinEdit;
    Label9: TLabel;
    RecStopVolume: TRadioButton;
    RecStopThreshold: TRadioButton;
    Label10: TLabel;
    Label11: TLabel;
    Label12: TLabel;
    RecModelStart: TSpinEdit;
    RecModelStop: TSpinEdit;
    RecModelInc: TSpinEdit;
    RecLoadReturn: TSpeedButton;
    RecShowTrainPerf: TCheckBox;
    Label13: TLabel;
    RecSetUpdateFreq: TSpinEdit;
    procedure RecPerformEarlyStopClick(Sender: TObject);
    procedure RecPerformModelClick(Sender: TObject);
    procedure RecUseGradDescentClick(Sender: TObject);
    procedure RecUseRPROPClick(Sender: TObject);
    procedure RecLoadReturnClick(Sender: TObject);
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
    procedure FormShow(Sender: TObject);
```

```
private
```

```
{ Private declarations }
maxepochs : integer;           // maximum number of epochs
numhidden : integer;           // number of hidden layer neurons
deltainit : real;               // initial delta values for RPROP
deltamax : real;               // maximum delta values for RPROP
learnrate : real;              // learning rate for gradient descent
momentum : real;               // momentum constant for gradient descent
numfails : integer;            // maximum number of fails before stop
                                // training if using early stopping
```

```
public
```

```
{ Public declarations }
function GetMaxEpochs : integer; // data access procedures
function GetNumHidden : integer;
function GetDeltaInit : real;
function GetDeltaMax : real;
function GetLearnRate : real;
function GetMomentum : real;
function GetNumFails : integer;
function GetUpdateFreq : integer;
```

```

end;

var
  RecSetParam: TRecSetParam;

implementation

uses loadunit, mainunit;

{$R *.DFM}

(* ----- DATA ACCESS PROCEDURES ----- *)
function TRecSetParam.GetMaxEpochs: integer;
begin
  GetMaxEpochs := maxepochs;
end;

function TRecSetParam.GetDeltainit: real;
begin
  GetDeltainit := deltainit;
end;

function TRecSetParam.GetDeltaMax: real;
begin
  GetDeltaMax := deltamax;
end;

function TRecSetParam.GetLearnRate: real;
begin
  GetLearnRate := learnrate;
end;

function TRecSetParam.GetMomentum: real;
begin
  GetMomentum := momentum;
end;

function TRecSetParam.GetNumFails: integer;
begin
  GetNumFails := numfails;
end;

function TRecSetParam.GetNumHidden: integer;
begin
  GetNumHidden := numhidden;
end;

function TRecSetParam.GetUpdateFreq: integer;
begin
  GetUpdateFreq := RecSetUpdateFreq.Value;
end;

(* ----- GENERAL PROCEDURES ----- *)
(* if user selects to perform early stopping, then enable user to set the
early stopping parameters *)
procedure TRecSetParam.RecPerformEarlyStopClick(Sender: TObject);
begin
  if RecPerformEarlyStop.Checked then
    RecSetEarlyParam.Enabled := true
  else
    RecSetEarlyParam.Enabled := false;
end;

(* if user selects to conduct a model search to determine the optimal number
of hidden layer neurons, then enable the user to set the model search parameters *)
procedure TRecSetParam.RecPerformModelClick(Sender: TObject);
begin
  if RecPerformModel.Checked then
    RecSetModelParam.Enabled := true
  else
    RecSetModelParam.Enabled := false;
end;

(* if user selects to use gradient descent, then enable the user to set the
training parameters specific to gradient descent *)
procedure TRecSetParam.RecUseGradDescentClick(Sender: TObject);
begin
  if RecUseGradDescent.Checked then
    RecRPropParam.Visible := false
  else
    RecRPropParam.Visible := true;

```

```

end;

(* if user selects to use Resilient back-propagation, then enable the user to
set the training parameters specific to RPROP *)
procedure TRecSetParam.RecUseRPropClick(Sender: TObject);
begin
  if RecUseRProp.Checked then
    RecRPropParam.Visible := true
  else
    RecRPropParam.Visible := false;
end;

(* close form *)
procedure TRecSetParam.RecLoadReturnClick(Sender: TObject);
begin
  Close;
end;

(* when close form, save the current network parameters to the registry and
dynamically allocate the memory for the different weight matrices*)
procedure TRecSetParam.FormClose(Sender: TObject;
  var Action: TCloseAction);
var Reg : TRegistry;
begin
  // save the current parameters to the registry
  Reg := TRegistry.Create;
  try
    Reg.OpenKey('Software\Neural', True);
    Reg.WriteInteger('MaxEpochs', strtoint(RecSetMaxEpochs.Text));
    Reg.WriteInteger('NumHidden', RecSetNumHidden.Value);
    Reg.WriteBool('EarlyStop', RecPerformEarlyStop.Checked);
    Reg.WriteBool('Model', RecPerformModel.Checked);
    Reg.WriteBool('UseGradient', RecUseGradDescent.Checked);
    Reg.WriteFloat('Deltainit', strtfloat(RecSetDeltainit.Text));
    Reg.WriteFloat('DeltaMax', strtfloat(RecSetDeltaMax.Text));
    Reg.WriteFloat('LearnRate', strtfloat(RecSetLearn.Text));
    Reg.WriteFloat('Momentum', strtfloat(RecSetMomentum.Text));
    Reg.WriteInteger('NumFails', RecNumFails.Value);
    Reg.WriteBool('StopVolume', RecStopVolume.Checked);
    Reg.WriteInteger('ModelStart', RecModelStart.Value);
    Reg.WriteInteger('ModelStop', RecModelStop.Value);
    Reg.WriteInteger('ModelInc', RecModelInc.Value);
    Reg.WriteBool('ShowTrainPerf', RecShowTrainPerf.Checked);
    Reg.WriteInteger('UpdateFreq', RecSetUpdateFreq.Value);
  finally
    Reg.Free;
  end;

  // get the required values from the form
  deltainit := strtfloat(RecSetDeltainit.Text);
  deltamax := strtfloat(RecSetDeltaMax.Text);
  learnrate := strtfloat(RecSetLearn.Text);
  momentum := strtfloat(RecSetMomentum.Text);
  numfails := RecNumFails.Value;

  if RecPerformModel.Checked then
    numhidden := RecModelStop.Value
  else
    numhidden := RecSetNumHidden.Value;

  maxepochs := strtoint(RecSetMaxEpochs.Text);

  // dynamically allocate the memory for the different weight matrices
  // now that the number of hidden nodes and the network training
  // procedure has been specified
  if numhidden > 0 then
    begin
      SetLength(OutputWeight, numhidden+1, RecLoadDBase.GetNumOutputs);
      SetLength(HiddenWeight, 129, numhidden);
      SetLength(OutputHidden, RecLoadDBase.GetTotalTrain
        +RecLoadDBase.GetTotalTest, numhidden+1);
      SetLength(DiffHiddenWeight, 129, numhidden);
      SetLength(DiffOutputWeight, numhidden+1, RecLoadDBase.GetNumOutputs);
      if RecUseRProp.Checked then
        begin
          // allocate the memory for the matrices required by RPROP
          SetLength(OutputDeltaWeight, numhidden+1, RecLoadDBase.GetNumOutputs);
          SetLength(OldDiffOutputWeight, numhidden+1, RecLoadDBase.GetNumOutputs);
          SetLength(HiddenDeltaWeight, 129, numhidden);
          SetLength(OldDiffHiddenWeight, 129, numhidden);

```

```

end
else
begin
// allocate the memory for the matrices required by gradient
// descent
SetLength(OldOutputWeight,numhidden+1,RecLoadDBase.GetNumOutputs);
SetLength(OldHiddenWeight,129,numhidden);
end;
end
else
begin
SetLength(OutputWeight,129,RecLoadDBase.GetNumOutputs);
SetLength(DiffOutputWeight,129,RecLoadDBase.GetNumOutputs);
if RecUseRProp.Checked then
begin
// allocate the memory for the matrices required by RPROP
SetLength(OutputDeltaWeight,129,RecLoadDBase.GetNumOutputs);
SetLength(OldDiffOutputWeight,129,RecLoadDBase.GetNumOutputs);
end
else
begin
// allocate the memory for the matrix required by gradient
// descent
SetLength(OldOutputWeight,129,RecLoadDBase.GetNumOutputs);
end;
end;
RecMain.Loadnetworkweights.Enabled := true;
RecMain.Savenetworkweights.Enabled := true;
RecMain.Initialisenetworkweights1.Enabled := true;
end;

```

(* when show form, load the previously used network training parameters from the registry and initialise the important variables *)

```

procedure TRecSetParam.FormShow(Sender: TObject);
var Reg: TRegistry;
    KeyGood: Boolean;
begin
// load the previously used parameters from the registry
Reg := TRegistry.Create;
try
KeyGood := Reg.OpenKey('Software\Neural', False);
// if the key exists then read from the key, else it is the
// first time the program is running so do not attempt
// to read from the registry
if KeyGood then
begin
RecSetMaxEpochs.Text := inttostr(Reg.ReadInteger('MaxEpochs'));
RecSetNumHidden.Value := Reg.ReadInteger('NumHidden');
RecPerformEarlyStop.Checked := Reg.ReadBool('EarlyStop');
RecPerformModel.Checked := Reg.ReadBool('Model');
RecUseGradDescent.Checked := Reg.ReadBool('UseGradient');
RecSetDeltaInit.Text := floattostr(Reg.ReadFloat('DeltaInit'));
RecSetDeltaMax.Text := floattostr(Reg.ReadFloat('DeltaMax'));
RecSetLearn.Text := floattostr(Reg.ReadFloat('LearnRate'));
RecSetMomentum.Text := floattostr(Reg.ReadFloat('Momentum'));
RecNumFails.Value := Reg.ReadInteger('NumFails');
RecStopVolume.Checked := Reg.ReadBool('StopVolume');
RecModelStart.Value := Reg.ReadInteger('ModelStart');
RecModelStop.Value := Reg.ReadInteger('ModelStop');
RecModelInc.Value := Reg.ReadInteger('ModelInc');
RecShowTrainPerf.Checked := Reg.ReadBool('ShowTrainPerf');
RecSetUpdateFreq.Value := Reg.ReadInteger('UpdateFreq');
end;
finally
Reg.Free;
end;

```

```

// set the important variables
if RecUseGradDescent.Checked then
    RecRPropParam.Visible := false
else
    RecRPropParam.Visible := true;
RecUseRProp.Checked := not RecUseGradDescent.Checked;
RecStopThreshold.Checked := not RecStopVolume.Checked;

```

```

if RecPerformEarlyStop.Checked then
    RecSetEarlyParam.Enabled := true
else
    RecSetEarlyParam.Enabled := false;
if RecPerformModel.Checked then
    RecSetModelParam.Enabled := true

```

```

else
    RecSetModelParam.Enabled := false;

if RecLoadDBase.RecDesiredRecon.ItemIndex = 0 then
    RecStopThreshold.Visible := true
else
begin
    RecStopThreshold.Visible := false;
    RecStopVolume.Checked := true;
end;
deltainit := strtofloat(RecSetDeltaInit.Text);
deltamax := strtofloat(RecSetDeltaMax.Text);
learnrate := strtofloat(RecSetLearn.Text);
momentum := strtofloat(RecSetMomentum.Text);
numfails := RecNumFails.Value;
numhidden := RecSetNumHidden.Value;
maxepochs := strtoint(RecSetMaxEpochs.Text);

// initialise the dynamically allocated memory
OutputWeight := nil;
HiddenWeight := nil;
DiffOutputWeight := nil;
OldOutputWeight := nil;
HiddenWeight := nil;
DiffHiddenWeight := nil;
OldHiddenWeight := nil;
OutputDeltaWeight := nil;
HiddenDeltaWeight := nil;
OldDiffHiddenWeight := nil;
OldDiffOutputWeight := nil;
end;
end.

```

PROGRAM LISTING OF NETWORKUNIT.PAS

(* This unit is used for the training of the neural networks. A single-layer feed-forward neural network and a double-layer feed-forward neural network can be trained using this unit. The network weights can be adjusted using either gradient descent of Resilient back-propagation and early-stopping is provided to prevent the network from overfitting the training data, which would result in poor generalisation performance. A model search feature is also provided to determine the optimal number of hidden layer neurons in a double-layer network. *)

unit networkunit;

interface

uses

Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs, ExtCtrls, ComCtrls, StdCtrls, Buttons, Math;

type

```

TRecTrainNetwork = class(TForm)
    Panel1: TPanel;
    Panel2: TPanel;
    Label1: TLabel;
    RecNetworkStart: TSpeedButton;
    Label2: TLabel;
    RecNetworkEpoch: TLabel;
    RecNetworkProgressBar: TProgressBar;
    Label3: TLabel;
    RecTrainResults: TPanel;
    Label4: TLabel;
    Label5: TLabel;
    RecTrainThreshErr: TLabel;
    RecTrainVoidErr: TLabel;
    RecTestThreshErr: TLabel;
    RecTestVoidErr: TLabel;
    RecThreshLabel: TLabel;
    RecSumLabel: TLabel;
    RecModelResults: TPanel;
    Label6: TLabel;
    Label7: TLabel;
    Label8: TLabel;
    Label9: TLabel;
    RecModelBestPerf: TLabel;

```

```

RecModelNumBest: TLabel;
RecModelCurrent: TLabel;
ModelSearchTimer: TTimer;
RecNetworkReturn: TSpeedButton;
PauseTrain: TSpeedButton;
procedure RecNetworkReturnClick(Sender: TObject);
procedure FormCreate(Sender: TObject);
procedure RecNetworkStartClick(Sender: TObject);
procedure FormShow(Sender: TObject);
procedure ModelSearchTimerTimer(Sender: TObject);
procedure PauseTrainClick(Sender: TObject);
private
{ Private declarations }
changeplus : real;           // multiplication factor for delta values
                             // in the RPROP algorithm (increase)
changeminus : real;         // multiplication factor for delta values
                             // in the RPROP algorithm (decrease)
numoutputs : integer;        // number of network outputs
numhidden : integer;         // number of hidden layer neurons
learnrate : real;            // learning rate for gradient descent
momentum : real;             // momentum constant for gradient descent
totalwaterpixels : real;     // number of test water pixels
totalgravelpixels : real;    // number of test gravel pixels
totalairpixels : real;       // number of test air pixels
totaltrainwaterpixels : real; // number of train water pixels
totaltraingravelpixels : real; // number of train gravel pixels
totaltrainairpixels : real;  // number of train air pixels
containsgravel : Boolean;    // whether contains gravel
containsair : Boolean;       // whether contains air
totaltrain : integer;        // total number of training cases
totaltest : integer;         // total number of test cases
maxepochs : integer;         // maximum number of epochs
maxfails : integer;          // maximum number of fails before stop
                             // network training if using early-
                             // stopping
updatefreq : integer;        // rate at which display updates
earlystop : Boolean;         // perform early-stopping
useRPROP : Boolean;          // train using RPROP
currtresh : real;            // current test threshold error
currsun : real;              // current test sum void error
currtrainthresh : real;      // current train threshold error
currtrainsum : real;         // current train sum void error
maxdelta : real;             // maximum delta value
mindelta : real;             // minimum delta value
watererror : real;           // current test water volume error
gravelerror : real;          // current test gravel volume error
airerror : real;             // current test air volume error
numbest : integer;           // number of hidden layer neurons for
                             // best performance
currentfails : integer;       // current number of fails
currepoch : integer;         // current training epoch
bestthresh : real;           // threshold corresponding to the best
                             // network
bestsum : real;              // sum void error corresponding to the
                             // best network
bestwater : real;            // water volume fraction corresponding
bestgravel : real;           // to the best network
bestair : real;
performingtrain : Boolean;
procedure TrainDoubleNetwork; // train double layer network
procedure TrainSingleNetwork; // train single layer network
procedure AdjustRProp;        // adjust network weights using RPROP
procedure AdjustWeights;      // adjust network weights using gradient
                             // descent
procedure CalculateTestOutput(start:integer;stop:integer);
                             // calculate the network outputs for the
                             // test data, as well as the validation
                             // error percentages
procedure CalculateTrainError(start:integer; stop:integer);
                             // calculate the training error
                             // percentages
function Sign(arg:extended):extended;
                             // return sign of arg
procedure UpdateNetworkDisplay; // update display during training
procedure StoreBestWeights;     // keep record of best network weights
                             // so far
public
{ Public declarations }
function SoftMax(numout:integer;wat:double;grav:double;air:double):double;
                             // calculate the soft-max activation
                             // function output

```

```

function CalcDeriv(a:extended):extended;
                             // calculate the derivative of the
                             // hyperbolic tangent
function GetAirError:real;    // data access functions
function GetWaterError:real;
function GetGravelError:real;
function GetSumVoidError:real;
function GetThresholdError:real;
function GetNumBest:integer;
end;

var
RecTrainNetwork: TRecTrainNetwork;

implementation

uses paramunit, loadunit, mainunit;

{$R *.DFM}

(* ----- NETWORK TRAINING PROCEDURES ----- *)
(* when start the network training, initialise all the important variables and
call the appropriate training procedure *)
procedure TRecTrainNetwork.RecNetworkStartClick(Sender: TObject);
begin
if not performingtrain then
begin
// initialise important variables
performingtrain := true;
RecNetworkStart.Caption := 'Stop training...';
RecMain.Savenetworkperformance1.Enabled := true;
RecMain.Validatedatabase1.Enabled := true;
RecMain.Validatecurrentsamples1.Enabled := true;
RecNetworkProgressBar.Position := 0;
numoutputs := RecLoadDBase.GetNumOutputs;
totaltrain := RecLoadDBase.GetTotalTrain;
totaltest := RecLoadDBase.GetTotalTest;
maxepochs := RecSetParam.GetMaxEpochs;
maxfails := RecSetParam.GetNumFails;
updatefreq := RecSetParam.GetUpdateFreq;
earlystop := RecSetParam.RecPerformEarlyStop.Checked;
useRPROP := RecSetParam.RecUseRPROP.Checked;
numhidden := RecSetParam.RecSetNumHidden.Value;
RecLoadDBase.GetPhases(containsair,containsgravel);
RecNetworkProgressBar.Max := RecSetParam.GetMaxEpochs div updatefreq;

if useRProp then
begin
maxdelta := RecSetParam.GetDeltaMax;
mindelta := 0;
end
else
begin
learnrate := RecSetParam.GetLearnRate;
momentum := RecSetParam.GetMomentum;
end;

if RecSetParam.RecPerformModel.Checked then
begin
// if wish to perform model search, then show the model
// search results panel and start the model search timer
numhidden := RecSetParam.RecModelStart.Value;
RecModelResults.Visible := true;
bestsum := 300;
bestthresh := 100;
ModelSearchTimer.Enabled := true;
PauseTrain.Visible := true;
end
else
begin
// else call the appropriate training procedure
if RecSetParam.GetNumHidden > 0 then
TrainDoubleNetwork
else
TrainSingleNetwork;
end;
performingtrain := false;
RecNetworkStart.Caption := 'Start training...';
end
else
begin

```

```

    currepoch := maxepochs;
    performingtrain := false;
    RecNetworkStart.Caption := 'Start training...';
end;
end;

(* procedure to train a single-layer feed-forward neural network where the
network weights can be modified using gradient descent or RPROP. Early stopping
is performed to prevent overfitting the training data thus ensuring good
generalisation performance for the test data *)
procedure TRecTrainNetwork.TrainSingleNetwork;
var outputnodeindex, inputnodeindex, currsample : integer;
    weightsum, newnodeoutput, newnoderiv, delta : real;
    weightstore : array[1..2] of real;
    i : integer;
begin
    // initialise important variables
    currepoch := 1;
    currentfails := 0;
    currthresh := 0;
    currsum := 0;

    // stop if the number of epochs exceeded or user specifies early stopping
    // and the number of fails has been exceeded i.e. the validation error
    // has continued to increase for a certain minimum number of iterations
    while (currepoch <= maxepochs) and (not((currentfails >= maxfails) and earllystop)) do
    begin
        // for each training point
        for currsample := 0 to totaltrain-1 do
        begin
            // for each output node calculate output of node
            for outputnodeindex := 0 to numoutputs-1 do
            begin
                weightsum := 0;
                // calculate the weighted summation to that node
                for inputnodeindex := 0 to 128 do
                begin
                    weightsum := weightsum + (InputData[currsample, inputnodeindex]
                        * OutputWeight[inputnodeindex, outputnodeindex]);
                end;

                // prevent arithmetic overflow
                if weightsum > 200 then weightsum := 200;
                else if weightsum < -200 then weightsum := -200;

                // if using 1-of-C then outputs can only be computed once the
                // weights to all three output nodes corresponding to a single
                // pixel have been found
                if numoutputs = 264 then
                begin
                    if ((outputnodeindex + 1) mod 3) <> 0 then
                        weightstore[(outputnodeindex+1) mod 3] := weightsum
                    else
                        begin
                            // we can now calculate the outputs of the SoftMax
                            // function since we have all three inputs to the
                            // nodes corresponding to water, gravel and air
                            // respectively
                            for i := 2 downto 0 do
                            begin
                                OutputNetwork[currsample, outputnodeindex-i] :=
                                    SoftMax(3-i, weightstore[1], weightstore[2], weightsum);

                                // calculate the adjustments to the output layer
                                // weights
                                delta := OutputNetwork[currsample, outputnodeindex-i]
                                    - OutputData[currsample, outputnodeindex-i];
                                for inputnodeindex := 0 to 128 do
                                begin
                                    DiffOutputWeight[inputnodeindex, outputnodeindex-i] :=
                                        DiffOutputWeight[inputnodeindex, outputnodeindex-i]
                                            + delta * InputData[currsample, inputnodeindex];
                                end;
                            end;
                        end;
                    end;
                end;
            end;
        end;
        // if wish to perform a regression then the output
        // activation function is linear
        if numoutputs = 2 then
        begin
            if weightsum >= 0 then

```

```

                newnodeoutput := weightsum
            else
                newnodeoutput := 0;
                newnoderiv := 1;
            end;
            // else just two phases so simple 'tanh' output
            else
            begin
                newnodeoutput := tanh(weightsum);
                newnoderiv := CalcDeriv(newnodeoutput);
            end;

            // calculate the adjustments to the output layer weights
            OutputNetwork[currsample, outputnodeindex] := newnodeoutput;
            delta := (newnodeoutput - OutputData[currsample, outputnodeindex]) *
                newnoderiv;

            for inputnodeindex := 0 to 128 do
            begin
                DiffOutputWeight[inputnodeindex, outputnodeindex] :=
                    DiffOutputWeight[inputnodeindex, outputnodeindex]
                        + delta * InputData[currsample, inputnodeindex];
            end;
        end;
    end;
end;

// after all the training data has been processed update the weights
// where the method of update depends on the training technique being
// used
if useRPROP then
    AdjustRPROP
else
    AdjustWeights;

// every updatefreq epochs, update the training display to indicate
// the current performance of the network
if currepoch mod updatefreq = 0 then
    UpdateNetworkDisplay;
    inc(currepoch);
end;
RecNetworkProgressBar.Position := RecNetworkProgressBar.Max;
end;

(* procedure to train a double-layer feed-forward neural network where the
hidden layer neurons consist of simple hyperbolic tangent activation functions. *)
procedure TRecTrainNetwork.TrainDoubleNetwork;
var outputnodeindex, inputnodeindex, currsample, i : integer;
    weightsum, deltasum, deltahidden, newderiv : real;
    delta : array of real;
    hiddenderiv : array of real;
    weightstore : array[1..2] of double;
begin
    // initialise the important variables
    currepoch := 1;
    currentfails := 0;
    SetLength(delta, numoutputs);
    SetLength(hiddenderiv, numhidden+1);

    // stop when the maximum number of epochs has been exceeded or when
    // early-stopping enabled and the validation performance has continued to
    // deteriorate for a certain minimum number of epochs set by maxfails
    while (currepoch <= maxepochs) and (not((currentfails >= maxfails) and earllystop)) do
    begin
        // for all training points
        for currsample := 0 to totaltrain-1 do
        begin
            // calculate outputs for hidden layer where the hidden layer
            // simply consists of tanh activation functions
            for outputnodeindex := 0 to numhidden do
            begin
                weightsum := 0;
                if outputnodeindex = 0 then
                begin
                    // bias node in hidden layer of neurons
                    OutputHidden[currsample, outputnodeindex] := 1;
                    hiddenderiv[outputnodeindex] := 0;
                end
                else
                begin
                    // calculate the weighted summation on the input

```

```

// to the hidden layer nodes
for inputnodeindex := 0 to 128 do
    weightsum := weightsum
    +InputData[currnsample,inputnodeindex]*
    HiddenWeight[inputnodeindex,outputnodeindex-1];

// prevent arithmetic overflow
if weightsum > 200 then weightsum := 200
else if weightsum < -200 then weightsum := -200;

// calculate hidden layer outputs
OutputHidden[currnsample,outputnodeindex]:=tanh(weightsum);
hiddenderiv[outputnodeindex] :=
    CalcDeriv(OutputHidden[currnsample,outputnodeindex]);
end;
end;
// calculate network outputs and adjustments to output weights
for outputnodeindex := 0 to numoutputs - 1 do
begin
    weightsum := 0;
    // calculate the weighted summation of the hidden layer
    // outputs to the output node weights
    for inputnodeindex := 0 to numhidden do
        weightsum := weightsum
        +OutputHidden[currnsample,inputnodeindex]
        *OutputWeight[inputnodeindex,outputnodeindex];

// prevent arithmetic overflow
if weightsum > 200 then weightsum := 200
else if weightsum < -200 then weightsum := -200;

// if use a 1-of-C encoding then the three outputs corresponding
// to a single pixel can only be calculated once the weighted
// sums for all three nodes have been calculated since a soft-max
// activation function is used
if numoutputs = 264 then
begin
    if ((outputnodeindex+1) mod 3) <> 0 then
        weightstore[(outputnodeindex+1) mod 3] := weightsum
    else
    begin
        // we can now calculate the outputs from the
        // softmax function since all three weighted
        // sums are known
        for i := 2 downto 0 do
        begin
            OutputNetwork[currnsample,outputnodeindex-i] :=
                SoftMax(3-i,weightstore[1],weightstore[2],
                weightsum);

// calculate the adjustments to the weights of
// the output layer nodes
delta[outputnodeindex-i] :=
    OutputNetwork[currnsample,outputnodeindex-i] -
    OutputData[currnsample,outputnodeindex-i];
        for inputnodeindex := 0 to numhidden do
            DiffOutputWeight[inputnodeindex,outputnodeindex-i]:=
                DiffOutputWeight[inputnodeindex,outputnodeindex-i]+
                delta[outputnodeindex-i]
                *OutputHidden[currnsample,inputnodeindex];

        end;
    end;
end;
else
begin
    // if regression then simple linear output unit
    if numoutputs = 2 then
    begin
        if weightsum >= 0 then
            OutputNetwork[currnsample,outputnodeindex] := weightsum
        else
            OutputNetwork[currnsample,outputnodeindex] := 0;

        newderiv := 1;
    end
    // else simply two phases so use tanh
    else
    begin
        OutputNetwork[currnsample,outputnodeindex] :=
            tanh(weightsum);

        newderiv :=
            CalcDeriv(OutputNetwork[currnsample,outputnodeindex]);
    end;

// calculate delta for output units
// a record of the delta values is kept to allow for the
// calculation of the weight adjustment for the hidden
// layers
delta[outputnodeindex] :=
    (OutputNetwork[currnsample,outputnodeindex]-
    OutputData[currnsample,outputnodeindex])*newderiv;
// calculate weight adjustment for output units
for inputnodeindex := 0 to numhidden do
    DiffOutputWeight[inputnodeindex,outputnodeindex]:=
        DiffOutputWeight[inputnodeindex,outputnodeindex]+
        delta[outputnodeindex]
        *OutputHidden[currnsample,inputnodeindex];
end;
end;

// calculate delta for hidden layer nodes
for inputnodeindex := 0 to numhidden - 1 do
begin
    deltasum := 0;
    // calculate the delta for the hidden layer by adding the
    // products of the deltas for the output nodes connected to
    // this hidden node by the weight of the connection
    for outputnodeindex := 0 to numoutputs - 1 do
        deltasum := deltasum
        +OutputWeight[inputnodeindex+1,outputnodeindex]*
        delta[outputnodeindex];
    deltahidden := hiddenderiv[inputnodeindex+1]*deltasum;
    // calculate adjustments to hidden weights
    for outputnodeindex := 0 to 128 do
        DiffHiddenWeight[outputnodeindex,inputnodeindex] :=
            DiffHiddenWeight[outputnodeindex,inputnodeindex]+
            deltahidden*InputData[currnsample,outputnodeindex];
    // where the derivative of the error function with
    // respect to the weights is given by the calculated
    // delta for the hidden node multiplied by the
    // input to that node
    end;
end;
// once gone through entire training set, make the adjustments
// to the weights where the type of adjustment depends on the
// training algorithm being used
if UseRPROP then
    AdjustRPROP
else
    AdjustWeights;

// every updatefreq epochs, update the network display to indicate the
// current training results
if currepoch mod updatefreq = 0 then
    UpdateNetworkDisplay;
inc(currepoch);

end;
RecNetworkProgressBar.Position := RecNetworkProgressBar.Max;
end;

(* update the network display to show current network training results. This
procedure also tests the validation performance of the network and whether
the validation performance is deteriorating as a result of overfitting the
training data *)
procedure TRecTrainNetwork.UpdateNetworkDisplay;
var tempthresh, tempsum : real;
begin
    // if user wishes to view performance of network for the training data
    if RecSetParam.RecShowTrainPerf.Checked then
    begin
        CalculateTrainError(0,totaltrain-1);
        RecTrainThreshErr.Caption := floattostrf(currtrainthresh,ffGeneral,7,5);
        RecTrainVoidErr.Caption := floattostrf(currtrainsum,ffGeneral,7,5);
    end;

    // keep a record of the current threshold and sum void errors so as to
    // provide for a comparison when the new values are calculated
    tempthresh := currthresh;
    tempsum := currsum;
    // calculate the network outputs for the test data, as well as the new

```

```

// threshold and sum void error percentages
CalculateTestOutput(totaltrain,totaltrain+totaltest-1);

if numoutputs <> 2 then
begin
  RecTestThreshErr.Caption := floattostrf(currthresh,ffGeneral,7,5);
end;
RecTestVoidErr.Caption := floattostrf(currsum,ffGeneral,7,5);

if earllystop then
begin
  // if wish to perform early stopping, then check whether
  // the validation performance is deteriorating
  if RecSetParam.RecStopThreshold.Checked then
  begin
    if currthresh > tempthresh then
      inc(currentfails)
    else
      currentfails := 0;
  end
  else
  begin
    if currsum > tempsum then
      inc(currentfails)
    else
      currentfails := 0;
  end;
end;
RecNetworkEpoch.Caption := 'Current epoch: '+inttostr(currepoch);
RecNetworkProgressBar.Stepl;
Application.ProcessMessages;
end;

(* ----- SPECIAL FUNCTIONS ----- *)
(* softmax activation function for the 1-of-C output encoding. This ensures that
the sum of the three dummy output variables always equals 1 and are therefore
valid posterior probabilities *)
function TRecTrainNetwork.SoftMax(numout:integer;wat:double;
                                grav:double;air:double):double;
var sum : double;
begin
  sum := exp(wat)+exp(grav)+exp(air);
  if numout = 1 then
    SoftMax := exp(wat)/sum
  else if numout = 2 then
    SoftMax := exp(grav)/sum
  else
    SoftMax := exp(air)/sum;
end;

(* calculate the derivative of the hyperbolic tangent at a specific point *)
function TRecTrainNetwork.CalcDeriv(a:extended):extended;
begin
  CalcDeriv := (1-intpower(a,2));
end;

(* return the sign of the argument *)
function TRecTrainNetwork.Sign(arg: extended): extended;
begin
  if arg > 0 then
    Sign := 1
  else if arg < 0 then
    Sign := -1
  else
    Sign := 0;
end;

(* ----- WEIGHT ADJUSTMENT PROCEDURES ----- *)
(* adjust the network weights using Resilient back-propagation *)
procedure TRecTrainNetwork.AdjustRProp;
var colindex, rowindex, indexend : integer;
begin
  indexend := 128;

  if numhidden > 0 then
  begin
    indexend := numhidden;
    // adjustments to hidden layer
    for colindex := 0 to numhidden -1 do
      begin
        for rowindex := 0 to 128 do

```

```

begin
  if OldDiffHiddenWeight[rowindex,colindex]*
  DiffHiddenWeight[rowindex,colindex] > 0 then
  begin
    // if same sign then increase amount by which change
    // the weights since 'going down hill'
    HiddenDeltaWeight[rowindex,colindex] :=
    Min((HiddenDeltaWeight[rowindex,colindex]*
    changeplus),maxdelta);
    HiddenWeight[rowindex,colindex] :=
    HiddenWeight[rowindex,colindex] - Sign(
    DiffHiddenWeight[rowindex,colindex])*
    HiddenDeltaWeight[rowindex,colindex];
    OldDiffHiddenWeight[rowindex,colindex] :=
    DiffHiddenWeight[rowindex,colindex];
  end
  else if OldDiffHiddenWeight[rowindex,colindex]*
  DiffHiddenWeight[rowindex,colindex] < 0 then
  begin
    // if opposite sign then crossed 'valley' so
    // reduce the amount by which change the weights
    // and return without making adjustment to weights
    HiddenDeltaWeight[rowindex,colindex] :=
    Max((HiddenDeltaWeight[rowindex,colindex]*
    changeminus),mindelta);
    OldDiffHiddenWeight[rowindex,colindex] := 0;
  end
  else
  begin
    // previous step crossed a 'valley' so can now adjust
    // weights
    HiddenWeight[rowindex,colindex] :=
    HiddenWeight[rowindex,colindex] -
    Sign(DiffHiddenWeight[rowindex,colindex])*
    HiddenDeltaWeight[rowindex,colindex];
    OldDiffHiddenWeight[rowindex,colindex] :=
    DiffHiddenWeight[rowindex,colindex];
  end;
  DiffHiddenWeight[rowindex,colindex] := 0;
end;
end;

end;

// adjustments to output layer
for colindex := 0 to numoutputs-1 do
begin
  for rowindex := 0 to indexend do
  begin
    if OldDiffOutputWeight[rowindex,colindex]*
    DiffOutputWeight[rowindex,colindex] > 0 then
    begin
      // same sign so increase amount by which change the
      // weights since 'going down hill'
      OutputDeltaWeight[rowindex,colindex] :=
      Min((OutputDeltaWeight[rowindex,colindex]*
      changeplus),maxdelta);
      OutputWeight[rowindex,colindex] :=
      OutputWeight[rowindex,colindex] - Sign(
      DiffOutputWeight[rowindex,colindex])*
      OutputDeltaWeight[rowindex,colindex];
      OldDiffOutputWeight[rowindex,colindex] :=
      DiffOutputWeight[rowindex,colindex];
    end
    else if OldDiffOutputWeight[rowindex,colindex]*
    DiffOutputWeight[rowindex,colindex] < 0 then
    begin
      // if opposite sign then crossed 'valley' so reduce
      // the amount by which the weights are changes and return
      // without changing weights
      OutputDeltaWeight[rowindex,colindex] :=
      Max((OutputDeltaWeight[rowindex,colindex]*
      changeminus),mindelta);
      OldDiffOutputWeight[rowindex,colindex] := 0;
    end
    else
    begin
      // previously crossed 'valley' so can now make cahnges to
      // weights
      OutputWeight[rowindex,colindex] :=
      OutputWeight[rowindex,colindex] -

```



```

        Sign(DiffOutputWeight[rowindex,colindex])*
        OutputDeltaWeight[rowindex,colindex];
        OldDiffOutputWeight[rowindex,colindex] :=
        DiffOutputWeight[rowindex,colindex];
    end;
    DiffOutputWeight[rowindex,colindex] := 0;

end;
end;
end;

(* adjust the network weights using gradient descent *)
procedure TRecTrainNetwork.AdjustWeights;
var rowindex, colindex, upperindex : integer;
    tempweight : double;
begin
    upperindex := 128;
    if numhidden > 0 then
    begin
        upperindex := numhidden;
        // for a double layer network
        // update the hidden layer weights
        for rowindex := 0 to 128 do
        begin
            for colindex := 0 to numhidden - 1 do
            begin
                tempweight := HiddenWeight[rowindex,colindex];

                // essentially  $w(t+1) = w(t) + LR * dE/dw + MOM(w(t) - w(t-1))$ 
                // is the update rule
                HiddenWeight[rowindex,colindex] :=
                    HiddenWeight[rowindex,colindex]
                    - learnrate*DiffHiddenWeight[rowindex,colindex]
                    + momentum*(HiddenWeight[rowindex,colindex]-
                    OldHiddenWeight[rowindex,colindex]);

                OldHiddenWeight[rowindex,colindex] := tempweight;
                DiffHiddenWeight[rowindex,colindex] := 0;
            end;
        end;
    end;

    // update the output layer weights
    for rowindex := 0 to upperindex do
    begin
        for colindex := 0 to numoutputs - 1 do
        begin
            tempweight := OutputWeight[rowindex,colindex];

            // essentially  $w(t+1) = w(t) + LR * dE/dw + MOM(w(t) - w(t-1))$ 
            // is the update rule
            OutputWeight[rowindex,colindex] := OutputWeight[rowindex,colindex]
            - learnrate*DiffOutputWeight[rowindex,colindex]
            + momentum*(OutputWeight[rowindex,colindex]-
            OldOutputWeight[rowindex,colindex]);

            OldOutputWeight[rowindex,colindex] := tempweight;
            DiffOutputWeight[rowindex,colindex] := 0;
        end;
    end;
end;

(* ----- NETWORK VALIDATION PROCEDURES ----- *)
(* calculate the network outputs for the test data by propagating the voltages
forward through the network. At the same time, calculate the threshold and
sum void error for the testing database *)
procedure TRecTrainNetwork.CalculateTestOutput(start, stop: integer);
var count, rowindex, colindex, i, upperindex, totalerrors: integer;
    tempvalue, weightsum : real;
    weightstore: array[1..2] of real;
begin
    // initialise important variables
    upperindex := 128;
    currthresh := 0;
    currsum := 0;
    totalerrors := 0;
    watererror := 0;
    gravelerror := 0;
    airerror := 0;
    // if there is a hidden layer then propagate the inputs through
    // the network calculating first the outputs from the hidden layer

```

```

// and then based on these outputs calculate the final network
// outputs
for count := start to stop do
begin
    if numhidden > 0 then
    begin
        // calculate the outputs of the hidden layer neurons where the
        // hidden layer neurons are simple hyperbolic tangents
        upperindex := numhidden;
        for colindex := 0 to numhidden do
        begin
            if colindex = 0 then
                OutputHidden[count,colindex] := 1 // bias term
            else
            begin
                weightsum := 0;
                // calculate the weighted sum on the input to the neuron
                for rowindex := 0 to 128 do
                    weightsum := weightsum +
                        HiddenWeight[rowindex,colindex-1]*
                        InputData[count,rowindex];

                // prevent arithmetic overflow
                if weightsum > 200 then weightsum := 200;
                else if weightsum < -200 then weightsum := -200;

                // tanh activation function
                OutputHidden[count,colindex] := tanh(weightsum);
            end;
        end;
    end;

    // calculate network output
    totalgravelpixels := 0;
    totalairpixels := 0;
    for colindex := 0 to numoutputs - 1 do
    begin
        weightsum := 0;
        // calculate the weighted sum on the input to the output neuron
        for rowindex := 0 to upperindex do
        begin
            if numhidden > 0 then
                tempvalue := OutputHidden[count,rowindex]
            else
                tempvalue := InputData[count,rowindex];

            weightsum := weightsum + OutputWeight[rowindex,colindex]*
                tempvalue;

        end;
        // prevent arithmetic overflow
        if weightsum > 200 then weightsum := 200;
        else if weightsum < -200 then weightsum := -200;

        // if using 1-of-C then three outputs corresponding to a pixel can
        // only be calculated once all three weighted inputs for those three
        // outputs have been calculated since uses soft-max activation function
        if numoutputs = 264 then
        begin
            if ((colindex + 1) mod 3) <> 0 then
                weightstore[(colindex+1) mod 3] := weightsum
            else
            begin
                // calculate the dummy variable outputs
                for i := 2 downto 0 do
                    OutputNetwork[count,colindex-i] :=
                        SoftMax(3-i,weightstore[1],weightstore[2],weightsum);

                // interpret the network outputs to determine the pixel
                // phase and whether this prediction is correct or not.
                // Also keep a record of the total number of air and
                // gravel pixels
                if (OutputNetwork[count,colindex]
                    >= OutputNetwork[count,colindex-1])
                    and (OutputNetwork[count,colindex]
                    >= OutputNetwork[count,colindex-2]) then
                begin
                    // predicts pixel is air
                    totalairpixels := totalairpixels+1;
                    // but if not air then increment the number of
                    // errors
                    if (OutputData[count,colindex] <> 1) then

```

```

        inc(totalelerrors);

    end
    else if(OutputNetwork[count,colindex-2]
        >=OutputNetwork[count,colindex-1])
    and(OutputNetwork[count,colindex-2]
        >OutputNetwork[count,colindex])then
    begin
        // predicts pixel is water
        // but if not water then increment the number of
        // errors
        if (OutputData[count,colindex-2] <> 1) then
            inc(totalelerrors);
        end
    else
    begin
        // predicts pixel is gravel
        totalgravelpixels := totalgravelpixels + 1;
        // but if not gravel then increment the number of
        // errors
        if (OutputData[count,colindex-1] <> 1) then
            inc(totalelerrors);
        end;
    end
    // two-phase prediction
    else if numoutputs = 88 then
    begin
        OutputNetwork[count,colindex] := tanh(weightsum);
        // if network output is less than 0 then predicts that
        // the pixel is water
        if(OutputNetwork[count,colindex] >= 0)then
        begin
            if OutputData[count,colindex] <> 1 then
                inc(totalelerrors);

            if containsair then
                totalairpixels := totalairpixels+1
            else
                totalgravelpixels := totalgravelpixels+1;

            end
        else
        begin
            if OutputData[count,colindex] <> -1 then
                inc(totalelerrors);
            end;
        end
        // volume fraction prediction
    else
    begin
        if weightsum >= 0 then
            OutputNetwork[count,colindex] := weightsum
        else
            OutputNetwork[count,colindex] := 0;
        if colindex = 0 then
            totalairpixels := OutputNetwork[count,colindex]
        else
            totalgravelpixels := OutputNetwork[count,colindex];
        end;
    end;
    totalwaterpixels := 100-(totalairpixels+totalgravelpixels);
    // calculate the errors in the volume fraction predictions for this
    // test case and add to the previous error percentages
    watererror := watererror + Abs(VolumeData[count,2]-totalwaterpixels);
    gravelerror := gravelerror + Abs(VolumeData[count,1]-totalgravelpixels);
    airerror := airerror + Abs(VolumeData[count,0]-totalairpixels);
    end;
    // calculate the threshold and sum void errors for the entire testing
    // database
    currthresh := 100*totalelerrors / (88*(stop-start+1));
    currsum := (watererror + gravelerror + airerror)/(stop-start+1);
    watererror := watererror/(stop-start+1);
    gravelerror := gravelerror/(stop-start+1);
    airerror := airerror/(stop-start+1);
end;

(* calculate the threshold and sum void error percentages for the training
database *)
procedure TRecTrainNetwork.CalculateTrainError(start, stop: integer);

```

```

var rowindex, colindex, totalelerrors:integer;
begin
    // initialise the important variables
    currtrainthresh := 0;
    currtrainsum := 0;
    totalelerrors := 0;
    watererror := 0;
    gravelerror := 0;
    airerror := 0;

    for rowindex := start to stop do
    begin
        totaltraingravelpixels := 0;
        totaltrainairpixels := 0;
        colindex := 0;
        while (colindex < numoutputs) do
        begin
            // three-phase prediction
            if numoutputs = 264 then
            begin
                if(OutputNetwork[rowindex,colindex+2]
                    >=OutputNetwork[rowindex,colindex+1])
                and(OutputNetwork[rowindex,colindex+2]
                    >=OutputNetwork[rowindex,colindex])then
                begin
                    // predicts pixel is air
                    totaltrainairpixels := totaltrainairpixels+1;
                    // but if not air then increment number of
                    // errors
                    if (OutputData[rowindex,colindex+2] <> 1) then
                        inc(totalelerrors);
                    end
                else if(OutputNetwork[rowindex,colindex]
                    >=OutputNetwork[rowindex,colindex+1])
                and(OutputNetwork[rowindex,colindex]
                    >OutputNetwork[rowindex,colindex+2])then
                begin
                    // predicts pixel is water
                    // but if not water then increment the number
                    // of errors
                    if (OutputData[rowindex,colindex] <> 1) then
                        inc(totalelerrors);
                    end
                else
                begin
                    // predicts pixel is gravel
                    totaltraingravelpixels := totaltraingravelpixels+1;
                    // but if not gravel the increment the number
                    // of errors
                    if (OutputData[rowindex,colindex+1] <> 1) then
                        inc(totalelerrors);
                    end;
                    inc(colindex,3);
                end
            // two-phase prediction
            else if numoutputs = 88 then
            begin
                // if network output less than 0 then predicts
                // that pixel is water
                if(OutputNetwork[rowindex,colindex] >= 0)then
                begin
                    if OutputData[rowindex,colindex] <> 1 then
                        inc(totalelerrors);

                    if containsair then
                        totaltrainairpixels :=
                            totaltrainairpixels+1
                    else
                        totaltraingravelpixels :=
                            totaltraingravelpixels+1;
                    end
                else
                begin
                    if OutputData[rowindex,colindex] <> -1 then
                        inc(totalelerrors);
                    end;
                    inc(colindex);
                end
            // volume fraction prediction

```

```

else
begin
    if colindex = 0 then
        totaltrainairpixels :=
            OutputNetwork[rowindex,colindex]
    else
        totaltraingravelpixels :=
            OutputNetwork[rowindex,colindex];
        inc(colindex);
    end;
end;
// calculate the volume fraction predictions for this test and add
// to the previous error percentages
totaltrainwaterpixels := 100-(totaltrainairpixels
+totaltraingravelpixels);
watererror := watererror + Abs(VolumeData[rowindex,2]
-totaltrainwaterpixels);
gravelerror := gravelerror + Abs(VolumeData[rowindex,1]
-totaltraingravelpixels);
airerror := airerror + Abs(VolumeData[rowindex,0]
-totaltrainairpixels);

end;
// calculate the threshold and sum void error percentages for the complete
// training database
currtrainthresh := 100*totalerrors / (88*(stop-start+1));
currtrainsum := (watererror + gravelerror + airerror)/(stop-start+1);

end;

(* ----- MODEL SEARCH PROCEDURES ----- *)
(* conduct a model search to determine the optimal number of hidden layer neurons
in a double-layer network. This is achieved by continually training networks where
the number of hidden layer neurons is incremented after each network has been
trained. The current network performance is then compared to the best performance
so far and the records updated appropriately *)
procedure TRecTrainNetwork.ModelSearchTimerTimer(Sender: TObject);
begin
    ModelSearchTimer.Enabled := false;
    PauseTrain.Enabled := false;
    RecModelCurrent.Caption := inttostr(numhidden);
    RecNetworkProgressBar.Position := 0;
    RecMain.RecWeightOKClick(Sender);

    // train the network for the current number of hidden layer neurons
    TrainDoubleNetwork;
    // compare the current performance to the best achieved performance and
    // update the record of the best performance appropriately - keep a
    // record of the weights for the best network by calling StoreBestWeights
    if RecSetParam.RecStopVolume.Checked then
        begin
            if currsum < bestsum then
                begin
                    numbest := numhidden;
                    bestsum := currsum;
                    bestwater := watererror;
                    bestair := airerror;
                    bestgravel := gravelerror;
                    bestthresh := currthresh;
                    RecModelBestPerf.Caption := floattostrf(bestsum,ffGeneral,7,5);
                    RecModelNumBest.Caption := inttostr(numbest);
                    StoreBestWeights;
                end;
            end;
        end
    else
        begin
            if currthresh < bestthresh then
                begin
                    numbest := numhidden;
                    bestthresh := currthresh;
                    bestsum := currsum;
                    bestwater := watererror;
                    bestair := airerror;
                    bestgravel := gravelerror;
                    RecModelBestPerf.Caption := floattostrf(bestthresh,ffGeneral,7,5);
                    RecModelNumBest.Caption := inttostr(numbest);
                    StoreBestWeights;
                end;
            end;
        end
    // increment the number of hidden layer neurons
    numhidden := numhidden + RecSetParam.RecModelInc.Value;

    // if not at the maximum number of hidden layer neurons then restart
    // the process
    if numhidden <= RecSetParam.RecModelStop.Value then
        begin
            ModelSearchTimer.Enabled := true;
            PauseTrain.Enabled := true;
            RecMain.Savenetworkperformance1.Click;
            RecMain.Savenetworkpreprocessing1.Click;
            RecMain.Savenetworkweights.Click;
        end
    else
        PauseTrain.Visible := false;
    end;

    (* keep a record of the weights corresponding to the best network performance *)
    procedure TRecTrainNetwork.StoreBestWeights;
    var colindex,rowindex : integer;
    begin
        OutputWeightBest := nil;
        HiddenWeightBest := nil;
        SetLength(OutputWeightBest,numhidden+1,numoutputs);
        SetLength(HiddenWeightBest,129,numhidden);
        for colindex := 0 to numhidden - 1 do
            for rowindex := 0 to 128 do
                HiddenWeightBest[rowindex,colindex] :=
                    HiddenWeight[rowindex,colindex];
            for colindex := 0 to numoutputs - 1 do
                for rowindex := 0 to numhidden do
                    OutputWeightBest[rowindex,colindex] :=
                        OutputWeight[rowindex,colindex];
                end;
            end;
        end;

        (* pause the model search *)
        procedure TRecTrainNetwork.PauseTrainClick(Sender: TObject);
        begin
            if ModelSearchTimer.Enabled = true then
                begin
                    ModelSearchTimer.Enabled := false;
                    PauseTrain.Caption := 'Resume Training';
                end
            else
                begin
                    ModelSearchTimer.Enabled := true;
                    PauseTrain.Caption := 'Pause training';
                end;
            end;

        end;

        (* ----- GENERAL PROCEDURES ----- *)
        (* close the form *)
        procedure TRecTrainNetwork.RecNetworkReturnClick(Sender: TObject);
        begin
            Close;
        end;

        (* when create the form, initialise the multiplication factors for RPROP. These
        factors control the rate at which RPROP converges on a solution *)
        procedure TRecTrainNetwork.FormCreate(Sender: TObject);
        begin
            changeplus := 1.2;
            changeminus := 0.5;
        end;

        (* when show form, update the display depending on the type of training to be
        performed *)
        procedure TRecTrainNetwork.FormShow(Sender: TObject);
        begin
            RecTrainResults.Visible := RecSetParam.RecShowTrainPerf.Checked;
            PauseTrain.Enabled := true;
            PauseTrain.Visible := false;
            if RecLoadDBase.GetNumOutputs = 2 then
                begin
                    RecTestThreshErr.Visible := false;
                    RecTrainThreshErr.Visible := false;
                    RecThreshLabel.Visible := false;
                end
            else
                begin
                    RecTestThreshErr.Visible := true;
                    RecTrainThreshErr.Visible := true;
                    RecThreshLabel.Visible := true;
                end;
            end;
        end;
    end;
end;

```

```

end;

(* ----- DATA ACCESS PROCEDURES ----- *)
(* access the network training results - if a model search has been conducted
then return the results corresponding to the best network*)
function TRecTrainNetwork.GetAirError: real;
begin
    if RecSetParam.RecPerformModel.Checked then
        GetAirError := bestair
    else
        GetAirError := aerror;
end;

function TRecTrainNetwork.GetGravelError: real;
begin
    if RecSetParam.RecPerformModel.Checked then
        GetGravelError := bestgravel
    else
        GetGravelError := gravelerror;
end;

function TRecTrainNetwork.GetSumVoidError: real;
begin
    if RecSetParam.RecPerformModel.Checked then
        GetSumVoidError := bestsum
    else
        GetSumVoidError := currsum;
end;

function TRecTrainNetwork.GetThresholdError: real;
begin
    if RecSetParam.RecPerformModel.Checked then
        GetThresholdError := bestthresh
    else
        GetThresholdError := currthresh;
end;

function TRecTrainNetwork.GetWaterError: real;
begin
    if RecSetParam.RecPerformModel.Checked then
        GetWaterError := bestwater
    else
        GetWaterError := watererror;
end;

function TRecTrainNetwork.GetNumBest: integer;
begin
    GetNumBest := numbest;
end;

end.

```

PROGRAM LISTING OF TESTUNIT.PAS

(* This unit is used for testing the trained networks - the networks can be tested using either data in the test database, or on samples obtained from the data acquisition system. The reconstructions take place continuously at a rate which can be specified by the user. For the test database option, the program cycles through all the test cases in the test database. *)

```

unit testunit;

interface

uses
    Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
    Buttons, ExtCtrls, TeEngine, Series, TeeProcs, Chart, Tomo, StdCtrls, Math,
    Spin;

type
    TRecTest = class(TForm)
        Panel1: TPanel;
        Panel2: TPanel;
        RecTestReturn: TSpeedButton;
        RecDesiredLabel: TLabel;
        RecTestDesired: TTomo;
        RecPredictLabel: TLabel;
        RecTestNetwork: TTomo;

```

```

        RecTestVolume: TPanel;
        RecAirChart: TChart;
        Series1: TBarSeries;
        RecGravChart: TChart;
        Series2: TBarSeries;
        RecWaterChart: TChart;
        BarSeries1: TBarSeries;
        RecVoidLabel: TLabel;
        Label4: TLabel;
        Label5: TLabel;
        Label6: TLabel;
        RecVoidNoteLabel: TLabel;
        RecTestStop: TSpeedButton;
        RecTestTimer: TTimer;
        Label1: TLabel;
        RecTestFrameRate: TSpinEdit;
        StartCalibrate: TSpeedButton;
        procedure RecTestReturnClick(Sender: TObject);
        procedure RecTestStopClick(Sender: TObject);
        procedure FormShow(Sender: TObject);
        procedure RecTestTimerTimer(Sender: TObject);
        procedure FormClose(Sender: TObject; var Action: TCloseAction);
        procedure StartCalibrateClick(Sender: TObject);
    private
        { Private declarations }
        testdata : array of real; // input voltages for a specific test case
        testhiddenoutput : array of real; // output of hidden layer neurons for
        // test case
        testoutput : array of real; // network output for test case
        testindex : integer; // index into testing database
        notifvalue : cardinal; // notification value for sampling
        // hardware to indicate completion of
        // sampling
        curcalibrate : Boolean;
        procedure CalculateTestCase; // calculate network output for test case
        procedure PlotTestOutput; // display network output
        procedure ProcessResults; // process sampled results
    public
        { Public declarations }
        procedure WndProc(var TheMsg:TMessage);override;
        // override the default Windows message
        // handling procedure to capture message
        // indicating sampling completion
    end;

var
    RecTest: TRecTest;
    usecurrentsamples : Boolean; // use samples taken from the DAQ
    result : array [0..51200] of double; // array to store sampling results
    calibdata : array [1..25600] of double;
    calibval : array [1..128] of double; // calibration reference point
implementation

```

uses loadunit, mainunit, networkunit, paramunit, realunit;

(\$R *.DFM)

```

(* ----- TESTING PROCEDURES ----- *)
(* each time the timer overflows, start a new test. For the testing database,
this is simply copying a new test case from the testing database and performing
a reconstruction. For the sampling, instruct the sampling hardware to collect
a new frame of data *)
procedure TRecTest.RecTestTimerTimer(Sender: TObject);
var colindex : integer;
begin
    RecTestTimer.Interval := floor(1000*(1/RecTestFrameRate.Value));
    if not usecurrentsamples then
    begin
        // copy the test case from the testing database
        for colindex := 0 to 128 do
            testdata[colindex] := InputData[testindex,colindex];
        // perform a reconstruction
        CalculateTestCase;
        // display network prediction
        PlotTestOutput;
        RecVoidLabel.Caption := 'Volume fraction predictions for test case '+
            inttostr(testindex);
        RecPredictLabel.Caption := 'Network prediction for test case '+
            inttostr(testindex);
        RecDesiredLabel.Caption := 'Desired output for test case '+
            inttostr(testindex);
    end;
end;

```

```

    inc(testindex);
    if testindex=(RecLoadDBase.GetTotalTrain+RecLoadDBase.GetTotalTest)then
        testindex := RecLoadDBase.GetTotalTrain;
    end
    else
        // instruct hardware to capture a frame of data
        SampHardware.StartSample(1);
end;

(* calculate the network output for the test case by propagating the input
voltages through the network *)
procedure TRecTest.CalculateTestCase;
var rowindex, colindex, i, upperindex, numoutputs, numhidden:integer;
    tempvalue, weightsum : real;
    weightstore:array[1..2] of real;
begin
    upperindex := 128;
    numoutputs := RecLoadDBase.GetNumOutputs;
    numhidden := RecSetParam.GetNumHidden;
    SetLength(testoutput,numoutputs);

    // if there is a hidden layer then propagate the inputs through
    // the network calculating first the outputs from the hidden layer
    // and then based on these outputs calculate the final network
    // outputs
    if numhidden > 0 then
    begin
        upperindex := numhidden;
        SetLength(testhiddenoutput,numhidden+1);

        for colindex := 0 to numhidden do
        begin
            if colindex = 0 then
                testhiddenoutput[colindex] := 1 // bias term
            else
                begin
                    weightsum := 0;
                    for rowindex := 0 to 128 do
                        weightsum := weightsum + HiddenWeight[rowindex,colindex-1]*
                            testdata[rowindex];

                    // prevent arithmetic overflow
                    if weightsum > 200 then weightsum := 200
                    else if weightsum < -200 then weightsum := -200;

                    // tanh activation function
                    testhiddenoutput[colindex] := tanh(weightsum);
                end;
            end;
        end;

        // calculate the network output
        for colindex := 0 to numoutputs - 1 do
        begin
            weightsum := 0;
            for rowindex := 0 to upperindex do
            begin
                if numhidden > 0 then
                    tempvalue := testhiddenoutput[rowindex]
                else
                    tempvalue := testdata[rowindex];

                weightsum := weightsum + OutputWeight[rowindex,colindex]*
                    tempvalue;
            end;
            // prevent arithmetic overflow
            if weightsum > 200 then weightsum := 200
            else if weightsum < -200 then weightsum := -200;

            // if using 1-of-C then three outputs corresponding to a pixel can
            // only be calculated once all three weighted inputs for those three
            // outputs have been calculated since uses soft-max activation function
            if numoutputs = 264 then
            begin
                if ((colindex + 1) mod 3) <> 0 then
                    weightstore[(colindex+1) mod 3] := weightsum
                else
                    begin
                        for i := 2 downto 0 do
                            testoutput[colindex-i] := RecTrainNetwork.SoftMax(3-i,
                                weightstore[1],weightstore[2],weightsum);

```

```

                                end;
                                end
                                // two-phase image reconstruction
                                else if numoutputs = 88 then
                                    testoutput[colindex] := tanh(weightsum)
                                else
                                    // volume fraction prediction
                                    begin
                                        if weightsum >= 0 then
                                            testoutput[colindex] := weightsum
                                        else
                                            testoutput[colindex] := 0;
                                        end;
                                    end;
                                end;

                                end;

                                (* interpret the network outputs and display the results *)
                                procedure TRecTest.PlotTestOutput;
                                var tempoutputpred, tempoutputdes : TOutput;
                                col, row, index, top, bottom: integer;
                                tempphasepred, tempphasesdes : TBubblePhase;
                                containsair, containsgravel : Boolean;
                                begin
                                    if RecTestVolume.Visible then
                                    begin
                                        // display the volume fractions in bar charts where the desired
                                        // volume fraction is the bar on the right and the network prediction
                                        // is the bar on the left
                                        RecAirChart.Series[0].Clear;
                                        RecAirChart.Series[0].Addy(testoutput[0],",",clTeeColor);
                                        if not usecurrentsamples then
                                            RecAirChart.Series[0].Addy(OutputData[testindex,0],",",clTeeColor);
                                        RecGravChart.Series[0].Clear;
                                        RecGravChart.Series[0].Addy(testoutput[1],",",clTeeColor);
                                        if not usecurrentsamples then
                                            RecGravChart.Series[0].Addy(OutputData[testindex,1],",",clTeeColor);
                                        RecWaterChart.Series[0].Clear;
                                        RecWaterChart.Series[0].Addy(100-(testoutput[0]+testoutput[1]),",",clTeeColor);
                                        if not usecurrentsamples then
                                            RecWaterChart.Series[0].Addy(100-
                                                (OutputData[testindex,0]+OutputData[testindex,1]),",",clTeeColor);
                                    end
                                    else
                                    begin
                                        // display an image of the vessel cross-section
                                        index := 0;

                                        for col := 0 to 9 do
                                        begin
                                            for row := 0 to 9 do
                                            begin
                                                tempoutputdes[col,row] := outpipe;
                                                tempoutputpred[col,row] := outpipe;
                                            end;
                                        end;

                                        end;
                                        for col := 0 to 9 do
                                        begin
                                            case col of
                                                0: begin top := 2; bottom := 7; end;
                                                1: begin top := 1; bottom := 8; end;
                                                2..7: begin top := 0; bottom := 9; end;
                                                8: begin top := 1; bottom := 8; end;
                                                9: begin top := 2; bottom := 7; end;
                                            end;

                                            end;
                                            for row := top to bottom do
                                            begin
                                                tempphasepred := water;
                                                tempphasesdes := water;
                                                // if two-phase reconstruction
                                                if RecLoadDBase.GetNumOutputs = 88 then
                                                begin
                                                    RecLoadDBase.GetPhases(containsair,containsgravel);
                                                    // if network output less than 0 then pixel is
                                                    // water, else pixel is other phase
                                                    if testoutput[index] >= 0 then
                                                    begin
                                                        if containsair then
                                                            tempphasepred := air
                                                        else
                                                            tempphasepred := gravel;

```

```

end;
if OutputData[testindex,index] = 1 then
begin
    if containsair then
        tempphasesdes := air;
    else
        tempphasesdes := gravel;
    end;
    inc(index);
end
// else three-phase prediction
else
begin
    if(testoutput[index+2]>=testoutput[index+1])
    and(testoutput[index+2]>=testoutput[index])then
        // pixel is air
        tempphasepred := air;
    else if(testoutput[index]>=testoutput[index+1])
    and(testoutput[index]>testoutput[index+2])then
        // pixel is water
        tempphasepred := water;
    else
        tempphasepred := gravel;

    if OutputData[testindex,index] = 1 then
        tempphasesdes := water;
    else if OutputData[testindex,index+1] = 1 then
        tempphasesdes := gravel;
    else
        tempphasesdes := air;

        inc(index,3);
    end;
    tempoutputdes[col,row] := tempphasesdes;
    tempoutputpred[col,row] := tempphasepred;
end;
end;
RecTestDesired.Desired := tempoutputdes;
RecTestNetwork.Desired := tempoutputpred;
// force a repaint to show changes
if not usecurrentsamples then
    RecTestDesired.Repaint;
RecTestNetwork.Repaint;
end;
end;

```

(* on completion of sampling, process the results so that only the voltages from the desired system are obtained and then pre-process the data according to the same rules applied to the training database. Then calculate the network output and plot the network prediction. *)

procedure TRecTest.ProcessResults;

var col,row, index: integer;

begin

// data was captured as test case

if not curcalibrate then

begin

col := 0;

row := 0;

index := 1;

// sort the voltages depending on which is the desired system

if RecLoadDBase.GetTomoSystem = 1 then

col := col+8;

while (col <= 255) do

begin

testdata[index] := result[col];

inc(row);

inc(index);

if row = 8 then

begin

row := 0;

inc(col,9);

end

else

inc(col);

end;

testdata[0] := 1;

// remove the calibration offset if the network was trained using

// calibrated data

if RecLoadDBase.RecLoadCalibrate.Checked then

for col := 1 to 128 do

begin

testdata[col] := testdata[col]-calibval[col];

end;

// standardise the data by subtracting the mean and dividing by the

// standard deviation

for col := 1 to 128 do

begin

testdata[col] := (testdata[col]-meanrec[col])/stddevrec[col];

end;

// calculate the network output and plot the results

CalculateTestCase;

PlotTestOutput;

end

// else capturing a set of data to provide a calibration point so that

// tests can be offset according to this calibration

else

begin

curcalibrate := false;

col := 0;

row := 0;

index := 1;

// offset the data according to which system wish to perform

// calibration for

if RecLoadDBase.GetTomoSystem = 1 then

col := col+8;

while (col < 12800) do

begin

calibdata[index] := result[col];

inc(row);

inc(index);

if row = 8 then

begin

row := 0;

inc(col,9);

end

else

inc(col);

end;

for index := 1 to 128 do

calibval[index] := 0;

for index := 1 to 6400 do

begin

row := index mod 128;

if row = 0 then

row := 128;

calibval[row] := calibval[row] + calibdata[index];

end;

for index := 1 to 128 do

calibval[index] := calibval[index]/50;

end;

end;

(* ----- GENERAL PROCEDURES ----- *)

(* override the default Windows message handling procedure to capture the

sampling complete message - when sampling is complete call StopSampling and

process the results *)

procedure TRecTest.WndProc(var TheMsg:TMessage);

begin

if TheMsg.Msg = notifvalue then

begin

SampHardware.StopSample(result);

ProcessResults;

end

else

inherited WndProc(TheMsg);

end;

(* close the form *)

procedure TRecTest.RecTestReturnClick(Sender: TObject);

begin

```

    RecTestTimer.Enabled := false;
    Close;
end;

(* provide the option of stopping the testing by disabling the timer which
triggers the start of a new test *)
procedure TRecTest.RecTestStopClick(Sender: TObject);
begin
    if RecTestTimer.Enabled then
    begin
        RecTestTimer.Enabled := false;
        RecTestStop.Caption := 'Continue';
    end
    else
    begin
        RecTestTimer.Enabled := true;
        RecTestStop.Caption := 'Stop tests';
    end;
end;

end;

(* when show form, initialise the sampling hardware and start the timer *)
procedure TRecTest.FormShow(Sender: TObject);
begin
    curcalibrate := false;
    RecTestStop.Caption := 'Start tests';
    SetLength(testdata, 129);

    // set the form parameters according to the type of test being
    // conducted
    if RecLoadDBase.RecDesiredRecon.ItemIndex = 0 then
    begin
        Caption := 'Test the network image reconstruction performance';
        RecTestVolume.Visible := false;
    end
    else
    begin
        Caption := 'Test the network volume fraction prediction';
        RecTestVolume.Visible := true;
    end;

    if usecurrentsamples then
    begin
        RecVoidLabel.Caption := 'Volume fraction predictions:';
        RecPredictLabel.Caption := 'Network prediction:';
        testindex := 1;
        RecVoidNoteLabel.Visible := false;
        RecDesiredLabel.Visible := false;
        RecTestDesired.Visible := false;
        StartCalibrate.Visible := true;
    end
    else
    begin
        testindex := RecLoadDBase.GetTotalTrain;
        RecVoidNoteLabel.Visible := true;
        RecDesiredLabel.Visible := true;
        RecTestDesired.Visible := true;
        StartCalibrate.Visible := false;
    end;
    RecTestTimer.Interval := 10;
    // initialise the sampling hardware
    notifvalue := SampHardware.InitSample(1, Handle);
    // start the timer

end;

(* when close form, free memory allocated for testing *)
procedure TRecTest.FormClose(Sender: TObject; var Action: TCloseAction);
begin
    testdata := nil;
    testoutput := nil;
    testhiddenoutput := nil;
end;

(* capture set of 50 frames to provide calibration point corresponding
to the rig full of water *)
procedure TRecTest.StartCalibrateClick(Sender: TObject);
begin
    RecMain.Savecalibrationdata1.Enabled := true;
    curcalibrate := true;

```

```

    SampHardware.StartSample(50);
end;

end.

```

PROGRAM LISTING OF REALUNIT.PAS

(* This unit implements the real-time sampling, reconstruction and cross-correlation calculation that would be used in an on-line flow monitoring application. To begin a test, the user specifies which network he wishes to use for the top system and which network he wishes to use for the bottom system. The application then loads the network weights and pre-processing data for those networks and, based on the data in these files, sets up the application for the appropriate reconstruction, as well as hidden layers, etc. To calculate the velocity of the individual components, the separation between the two measurement systems must be provided as well as the flowmeter factor. Since the readings from the measurement systems contain ripple, the volume fraction predictions also contain ripple - this can be removed by specifying a threshold level larger than this noise level. Once the stepper motors have been configured, the test is started. After the specified number of frames has been captured and the reconstructions calculated, a cross-correlation on the gravel and air volume fractions is performed over the duration of these frames. By finding the peaks of the cross-correlation functions, the algorithm is able to determine the transit time between the two measurement systems and hence calculate the individual component velocities. *)

unit realunit;

interface

uses

Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs, Buttons, StdCtrls, ExtCtrls, Tomo, Spin, math, TeEngine, Series, TeeProcs, Chart, Registry, StepUnit;

type

```

TRealMain = class(TForm)
    Panel2: TPanel;
    Panel3: TPanel;
    Label1: TLabel;
    TopTomo: TTomo;
    BottomTomo: TTomo;
    Label2: TLabel;
    Label3: TLabel;
    Label4: TLabel;
    Label5: TLabel;
    Label6: TLabel;
    RealCalibrate: TSpeedButton;
    RealStart: TSpeedButton;
    Label9: TLabel;
    Label10: TLabel;
    persec: TLabel;
    permsec: TLabel;
    totframes: TLabel;
    Label11: TLabel;
    frmrate: TLabel;
    RealOnline: TRadioGroup;
    RealExit: TSpeedButton;
    RealWeights: TSpeedButton;
    Label7: TLabel;
    Label8: TLabel;
    BottomTestNum: TSpinEdit;
    TopTestNum: TSpinEdit;
    RealView: TSpeedButton;
    RealTimer: TTimer;
    RealFrameRate: TSpinEdit;
    Label12: TLabel;
    Label13: TLabel;
    frmnumber: TLabel;
    permin: TLabel;
    TopVolume: TPanel;
    BottomVolume: TPanel;
    TopAirChart: TChart;
    Series1: TBarSeries;
    TopGravChart: TChart;
    Series2: TBarSeries;
    TopWaterChart: TChart;
    BarSeries1: TBarSeries;

```

```

BottomAirChart: TChart;
BarSeries2: TBarSeries;
BottomGravChart: TChart;
BarSeries3: TBarSeries;
BottomWaterChart: TChart;
BarSeries4: TBarSeries;
Label14: TLabel;
Label15: TLabel;
Label16: TLabel;
Label17: TLabel;
Label18: TLabel;
Label19: TLabel;
Label20: TLabel;
DesiredFrames: TSpinEdit;
Label21: TLabel;
smins: TLabel;
ssecs: TLabel;
smsecs: TLabel;
MotorSetup: TSpeedButton;
StepperStop: TSpeedButton;
Motor1Setting: TLabel;
Motor2Setting: TLabel;
AirVelActual: TLabel;
GravelVelActual: TLabel;
RealStartMotors: TSpeedButton;
AirPlot: TChart;
Series3: TLineSeries;
Series4: TLineSeries;
GravelPlot: TChart;
LineSeries1: TLineSeries;
LineSeries2: TLineSeries;
AirCorrPlot: TChart;
LineSeries3: TLineSeries;
GravelCorrPlot: TChart;
LineSeries4: TLineSeries;
Label22: TLabel;
Label23: TLabel;
SystemSepInput: TEdit;
FlowFactorInput: TEdit;
Label24: TLabel;
Label25: TLabel;
Label26: TLabel;
AirVelPredict: TLabel;
GravelVelPredict: TLabel;
SetThreshold: TPanel;
Label27: TLabel;
TopAirThresh: TEdit;
Label28: TLabel;
Label29: TLabel;
TopGravThresh: TEdit;
Label30: TLabel;
BotAirThresh: TEdit;
Label31: TLabel;
BotGravThresh: TEdit;
Label32: TLabel;
CorrThreshInp: TEdit;
VerifyResults: TSpeedButton;
Label33: TLabel;
ViewInc: TSpinEdit;
procedure RealExitClick(Sender: TObject);
procedure FormShow(Sender: TObject);
procedure RealCalibrateClick(Sender: TObject);
procedure RealStartClick(Sender: TObject);
procedure RealWeightsClick(Sender: TObject);
procedure RealViewClick(Sender: TObject);
procedure RealTimerTimer(Sender: TObject);
procedure MotorSetupClick(Sender: TObject);
procedure StepperStopClick(Sender: TObject);
procedure RealStartMotorsClick(Sender: TObject);
procedure VerifyResultsClick(Sender: TObject);
private
{ Private declarations }
notifvalue : cardinal;           // Windows notification message
                                // value that is issued once
                                // streaming is complete

curcalibrate : Boolean;
topcalib : array[1..6400] of real; // calibration data for top
bottomcalib : array[1..6400] of real; // and bottom systems
topcalibvalue : array[1..128] of real;
bottomcalibvalue : array[1..128] of real;
bottommean : array[1..128] of real; // pre-processing data (loaded

bottomstddev : array[1..128] of real; // from a file) for the top
topmean : array[1..128] of real; // and bottom neural network
topstddev : array[1..128] of real;
topweights : array of array of real; // matrices containing neural
tophiddenweights : array of array of real; // network weights for the top
tophiddenoutputs : array of real; // and bottom system - both hidden
bottomhiddenoutputs : array of real; // layer and output layer
bottomweights : array of array of real;
bottomhiddenweights : array of array of real;
totalfrms : integer; // total number of frames to be
                        // captured
hours, mins, secs, msec : word; // used to calculate the total
                                // time taken to capture and
                                // reconstruct the specified
                                // number of frames
storeposition : integer; // position in matrix where data
                        // must be placed
viewposition : integer; // position in matrix where data
                        // being viewed
starttime : TDateTime; // time test started
stoptime : TDateTime; // time test finished
timediff : TDateTime; // time difference
numtophidden : integer;
numbottomhidden : integer;
busyreconstruction : Boolean; // indicate whether still busy
                                // performing a reconstruction
bussyampling : Boolean; // indicate whether still busy
                        // sampling
upperindex : integer;
airdelay : integer; // air transit time
graveldelay : integer; // gravel transit time
topairthreshold : real; // threshold values
topgravthreshold : real;
botairthreshold : real;
botgravthreshold : real;
correlationthreshold : real;
drive1steps : integer; // stepper motor parameters
drive2steps : integer;
drive3steps : integer;
drive1rps : real;
drive2rps : real;
drive3rps : real;
procedure ProcessResults;
procedure PerformCorrelation; // perform cross-correlation
procedure RealReconstruction; // perform reconstruction
procedure StepperInit; // initialise stepper
procedure PlotTestOutput(testposition : integer);
                        // display output
public
{ Public declarations }
Stepper : TStep; // object used to control and
                // access AT6400 stepper motor
                // controller
frameinterval : real; // estimated interval between
                    // frames
airvelocity : real;
gravelvelocity : real;
airactualvel : real;
gravelactualvel : real;
flowmeterfactor : real;
systemseparation : real;
procedure WndProc(var TheMsg:TMessage);override;
end;

TTomoInputData = record // when sample data, keep a record
    readings : array [1..256] of real; // of the system time as well for
    sampletime : TDateTime; // comparison
end;
TTomoOutputData = record
    prediction : array [1..528] of real; // when output data, copy the sample
    sampletime : TDateTime; // time across
end;
var
RealMain: TRealMain;
result : array [0..12800] of double; // temporary buffer for samples
inputstore : array[1..6000] of TTomoInputData; // once the sampled data has
                                                // been processed, it is placed in
                                                // the inputstore - the
                                                // reconstruction algorithm then
                                                // takes points from this store
                                                // and places the reconstructed

```



```

// outputs in the outputstore
outputstore : array[1..6000] of TTomoOutputData;
volumestore : array[1..6000,1..4] of real;
gravellcorr : array[1..6001] of real; // gravel and air correlation
aircorr : array[1..6001] of real; // results
recomposition : integer; // position in matrix where data
// is being taken from for
// reconstruction
totaloutputs : integer;

```

implementation

uses mainunit, networkunit, stepconfig, dynunit, loadunit;

{ \$R *.DFM }

(* ----- SAMPLING AND RECONSTRUCTION PROCEDURES ----- *)

(* initialise variables for test, start stepper motion and initiate the capture of 50 frames of data *)

procedure TRealMain.RealStartClick(Sender: TObject);

var i,tempsteps : integer;

begin

```

// initialise charts and other test variables
AirPlot.Series[0].Clear;
AirPlot.Series[1].Clear;
GravelPlot.Series[0].Clear;
GravelPlot.Series[1].Clear;
AirCorrPlot.Series[0].Clear;
GravelCorrPlot.Series[0].Clear;
topairthreshold := strtoint(TopAirThresh.Text);
topgravthreshold := strtoint(TopGravThresh.Text);
botairthreshold := strtoint(BotAirThresh.Text);
botgravthreshold := strtoint(BotGravThresh.Text);
systemseparation := strtoint(SystemSepInput.Text);
flowmeterfactor := strtoint(FlowFactorInput.Text);
correlationthreshold := strtoint(CorrThreshInp.Text);
for i := 1 to 6000 do
begin
    volumestore[i,1] := 0;
    volumestore[i,2] := 0;
    volumestore[i,3] := 0;
    volumestore[i,4] := 0;
end;

```

```

StepperStop.Visible := true;
VerifyResults.Visible := true;
busyreconstruction := false;
busysampling := true;
RealStart.Caption := 'Stop testing';
totalfrms := 1;
storeposition := 1;
recomposition := 1;
curcalibrate := false;

```

// determine the desired direction and speed of the stepper motors
// for both the air and gravel phases

StepperInit;

if StepperConfig.airrps > 0 then

begin

if StepperConfig.FlowDirSelect.ItemIndex = 0 then

begin

if StepperConfig.AirBubblePosition.ItemIndex = 0 then

tempsteps := StepperConfig.airsteps

else

tempsteps := -StepperConfig.airsteps;

end

else

begin

if StepperConfig.AirBubblePosition.ItemIndex = 0 then

tempsteps := -StepperConfig.airsteps

else

tempsteps := StepperConfig.airsteps;

end;

case StepperConfig.AirDrive.Value of

1: begin

drive1rps := StepperConfig.airrps;

drive1steps := tempsteps;

end;

2: begin

drive2rps := StepperConfig.airrps;

drive2steps := -tempsteps;

```

end;
3: begin
    drive3rps := StepperConfig.airrps;
    drive3steps := tempsteps;
end;
end;
end;

if StepperConfig.gravelrps > 0 then
begin
    if StepperConfig.FlowDirSelect.ItemIndex = 0 then
    begin
        if StepperConfig.GravelBubblePosition.ItemIndex = 0 then
        tempsteps := StepperConfig.gravelsteps
        else
        tempsteps := -StepperConfig.gravelsteps;
        end
        else
        begin
            if StepperConfig.GravelBubblePosition.ItemIndex = 0 then
            tempsteps := -StepperConfig.gravelsteps
            else
            tempsteps := StepperConfig.gravelsteps;
            end;
        end;

        case StepperConfig.GravelDrive.Value of
        1: begin
            drive1rps := StepperConfig.gravelrps;
            drive1steps := tempsteps;
            end;
        2: begin
            drive2rps := StepperConfig.gravelrps;
            drive2steps := -tempsteps;
            end;
        3: begin
            drive3rps := StepperConfig.gravelrps;
            drive3steps := tempsteps;
            end;
        end;
    end;
end;
// start the stepper motion
Stepper.MoveFixedDist(drive1rps,drive2rps,drive3rps,drive1steps,
    drive2steps,drive3steps);
// keep a record of when started sampling
starttime := Now;
// initiate the capture of 50 frames of data
SampHardware.StartSample(50);
end;

```

(* the real-time unit operates as follows:

1. data is captured in sets of 50 frames and is then separated and pre-processed for the top and bottom system
 2. initially 50 frames of data is captured
 3. once the sampling is complete, the system is instructed to capture another set of 50 frames
 4. while the sampling is in process, the reconstructions for the previous set of 50 frames is performed
 5. busyreconstruction and busysampling provide flags indicating when it is OK to start the capture of the next set of 50 frames or perform the reconstructions for the previous set of frames
 6. if the application is still busy with the reconstruction for the previous set of frames then the sampling stops until it is complete (otherwise overflow)
- In this way, the sampling and reconstruction tasks are overlapped thus achieving maximum possible frame rate for the system. *)

procedure TRealMain.ProcessResults;

var index, topindex, bottomindex, cnt : integer;

topsyst : Boolean;

begin

// pre-process for a calibration

if curcalibrate = true then

begin

curcalibrate := false;

topindex := 1;

bottomindex := 1;

topsyst := true;

cnt := 0;

// separate the data between the two systems

for index := 0 to 12799 do

begin

if topsyst then

begin

```

        topcalib[topindex] := result[index];
        inc(topindex);
    end
else
begin
    bottomcalib[bottomindex] := result[index];
    inc(bottomindex);
end;
inc(cnt);
if cnt = 8 then
begin
    topsyst := not topsyst;
    cnt := 0;
end;
end;

for index := 1 to 128 do
begin
    topcalibvalue[index] := 0;
    bottomcalibvalue[index] := 0;
end;

for index := 1 to 6400 do
begin
    cnt := index mod 128;
    if cnt = 0 then
        cnt := 128;

        topcalibvalue[cnt] := topcalibvalue[cnt]+topcalib[index];
        bottomcalibvalue[cnt] := bottomcalibvalue[cnt]+
            bottomcalib[index];
    end;

    // find the mean of the different inputs over the 50 frames
    for index := 1 to 128 do
    begin
        topcalibvalue[index] := topcalibvalue[index]/50;
        bottomcalibvalue[index] := bottomcalibvalue[index]/50;
    end;
end
// else real-time captured data
else
begin
    topindex := 1;
    bottomindex := 1;
    topsyst := true;
    cnt := 0;
    // pre-process the data by performing calibration and standardisation
    // and place the data in inputstore
    for index := 0 to 12799 do
    begin
        if topsyst then
        begin
            inputstore[storeposition].readings[topindex] :=
                (result[index] - topcalibvalue[topindex]
                -topmean[topindex])/topstddev[topindex];

            inc(topindex);
            if topindex = 129 then
                topindex := 1;
        end
        else
        begin
            inputstore[storeposition].readings[bottomindex + 128] :=
                (result[index] - bottomcalibvalue[bottomindex]
                -bottommean[bottomindex])/bottomstddev[bottomindex];
            inc(bottomindex);
            if bottomindex = 129 then
            begin
                bottomindex := 1;
                inputstore[storeposition].sampletime := Now-starttime;
                inc(storeposition);
                if storeposition > 6000 then
                    storeposition := 1;
                inc(totalfrms);
            end;
        end;
        inc(cnt);
        if cnt = 8 then
            begin
                topsyst := not topsyst;

```

```

        cnt := 0;
    end;
end;
busysampling := false;
// if still more frames to capture
if (totalfrms < DesiredFrames.Value) then
begin
    // if finished with the previous reconstruction then
    // start the capture of another set of frames and perform the
    // reconstruction of this set
    if not busyreconstruction then
    begin
        busysampling := true;
        SampHardware.StartSample(50);
        busyreconstruction := true;
        RealReconstruction;
    end;
end
else
begin
    // do the reconstruction for the final set of 50 frames
    RealReconstruction;
    RealStart.Caption := 'Start testing';
    StepperStop.Visible := false;
    stoptime := Now;
    timediff := stoptime - starttime;
    // calculate the frame rate
    DecodeTime(timediff, hours, mins, secs, msecs);
    permin.Caption := intostr(mins);
    persec.Caption := intostr(secs);
    permsec.Caption := intostr(msecs);
    totframes.Caption := intostr(totalfrms - 1);
    frameinterval := (mins*60+secs+msecs/1000)/(totalfrms - 1);
    frmrate.Caption := floattostr(1/frameinterval, ffFixed, 10, 6);
    Update;
    // perform a correlation on this captured data
    PerformCorrelation;
end;
end;
end;

(* this procedure implements the double-layer feedforward neural network
reconstruction on the real-time captured data and places the results in
outputstore - very similar to the reconstruction algorithm for the test unit*)
procedure TRealMain.RealReconstruction;
var rowindex, colindex, i: integer;
    tempvalue, weightsum : real;
    weightstore: array[1..2] of real;
    numrecon : integer;
begin
    for numrecon := 1 to 50 do
    begin
        // first the top rig
        upperindex := 128;

        // if there are hidden layer neurons then calculate the output
        // from the hidden layer first
        if numtophidden > 0 then
        begin
            upperindex := numtophidden;
            for colindex := 0 to numtophidden do
            begin
                if colindex = 0 then
                    tophiddenoutputs[colindex] := 1
                else
                begin
                    weightsum := 0;
                    for rowindex := 0 to 128 do
                    begin
                        if rowindex = 0 then
                            tempvalue := 1
                        else
                            tempvalue := inputstore[reconposition].readings[rowindex];

                        weightsum := weightsum + tophiddenweights[rowindex, colindex-1]*
                            tempvalue;
                    end;

                    // prevent arithmetic overflow
                    if weightsum > 200 then weightsum := 200
                    else if weightsum < -200 then weightsum := -200;

```

```

    // tanh activation function
    tophiddenoutputs[colindex] := tanh(weightsum);
end;
end;
end;

// once calculated hidden layer outputs, calculate final network
// prediction
for colindex := 0 to totaloutputs-1 do
begin
    weightsum := 0;
    for rowindex := 0 to upperindex do
    begin
        if numtophidden = 0 then
        begin
            if rowindex = 0 then
                tempvalue := 1
            else
                tempvalue := inputstore[recomposition].readings[rowindex];
            end
        else
            tempvalue := tophiddenoutputs[rowindex];
        end

        weightsum := weightsum + topweights[rowindex,colindex]*
            tempvalue;
    end;
    if weightsum > 200 then weightsum := 200
    else if weightsum < -200 then weightsum := -200;
    // three-phase image reconstruction
    if totaloutputs = 264 then
    begin
        if ((colindex + 1) mod 3) <> 0 then
            weightstore[(colindex+1) mod 3] := weightsum
        else
        begin
            for i := 2 downto 0 do
                outputstore[recomposition].prediction[colindex+i+1] :=
                    RecTrainNetwork.SoftMax(3-i,weightstore[1],weightstore[2],weightsum);
            end;
        end
        // two-phase image reconstruction
        else if totaloutputs = 88 then
        begin
            outputstore[recomposition].prediction[colindex+1] := tanh(weightsum);
        end
        // three-phase volume fraction prediction
        else
        begin
            if (colindex+1) = 1 then
            begin
                if weightsum >= topairthreshold then
                    outputstore[recomposition].prediction[1] := weightsum
                else
                    outputstore[recomposition].prediction[1] := 0;
                end
            else
            begin
                if weightsum >= topgravthreshold then
                    outputstore[recomposition].prediction[2] := weightsum
                else
                    outputstore[recomposition].prediction[2] := 0;
                end;
            end;
        end;
    end;
end;

// now the bottom rig
upperindex := 128;

// if there are hidden layer neurons then calculate the output
// from the hidden layer first
if numbottomhidden > 0 then
begin
    upperindex := numbottomhidden;
    for colindex := 0 to numbottomhidden do
    begin
        if colindex = 0 then
            bottomhiddenoutputs[colindex] := 1
        else
        begin
            weightsum := 0;

```

```

        for rowindex := 0 to 128 do
        begin
            if rowindex = 0 then
                tempvalue := 1
            else
                tempvalue := inputstore[recomposition].readings[rowindex+128];

            weightsum := weightsum + bottomhiddenweights[rowindex,colindex-1]*
                tempvalue;
        end;

        // prevent arithmetic overflow
        if weightsum > 200 then weightsum := 200
        else if weightsum < -200 then weightsum := -200;

        // tanh activation function
        bottomhiddenoutputs[colindex] := tanh(weightsum);
    end;
end;
end;

// once calculated hidden layer outputs, calculate the final
// network prediction
for colindex := 0 to totaloutputs - 1 do
begin
    weightsum := 0;
    for rowindex := 0 to upperindex do
    begin
        if numbottomhidden = 0 then
        begin
            if rowindex = 0 then
                tempvalue := 1
            else
                tempvalue := inputstore[recomposition].readings[rowindex+128];
            end
        else
            tempvalue := bottomhiddenoutputs[rowindex];
        end

        weightsum := weightsum + bottomweights[rowindex,colindex]*
            tempvalue;
    end;
    if weightsum > 200 then weightsum := 200
    else if weightsum < -200 then weightsum := -200;
    // three-phase image reconstruction
    if totaloutputs = 264 then
    begin
        if ((colindex + 1) mod 3) <> 0 then
            weightstore[(colindex+1) mod 3] := weightsum
        else
        begin
            for i := 2 downto 0 do
                outputstore[recomposition].prediction[colindex+i+265] :=
                    RecTrainNetwork.SoftMax(3-i,weightstore[1],weightstore[2],weightsum);
            end;
        end
        // two-phase image reconstruction
        else if totaloutputs = 88 then
        begin
            outputstore[recomposition].prediction[colindex+89] := tanh(weightsum);
        end
        // three-phase volume fraction prediction
        else
        begin
            if (colindex+3) = 3 then
            begin
                if weightsum >= botairthreshold then
                    outputstore[recomposition].prediction[3] := weightsum
                else
                    outputstore[recomposition].prediction[3] := 0;
                end
            else
            begin
                if weightsum >= botgravthreshold then
                    outputstore[recomposition].prediction[4] := weightsum
                else
                    outputstore[recomposition].prediction[4] := 0;
                end;
            end;
        end;
    end;
end;

// copy the sample time across from the inputstore to keep as record for

```

```

// analysis
outputstore[recomposition].sampletime := inputstore[recomposition].sampletime;
// if wish to display the results online, then update the display
if (RealOnline.ItemIndex = 0) then
begin
    PlotTestOutput(recomposition);
end;
inc(recomposition);

end;

if recomposition > 6000 then
    recomposition := 1;

busyreconstruction := false;

// if sampling was stopped to allow for the reconstruction to complete
// then restart the sampling and perform the reconstructions for the
// previous set of 50 frames
if (not busysampling)and(totalfrms < DesiredFrames.Value) then
begin
    busysampling := true;
    SampHardware.StartSample(50);
    if recomposition < (storeposition - 50) then
        RealReconstruction;
    end;
end;

(* display the real-time reconstruction results *)
procedure TRealMain.PlotTestOutput(testposition : integer);
var topoutputpred, bottomoutputpred : TOutput;
    topphasepred, bottomphasepred : TBubblePhase;
    index, col, row, top, bottom : integer;
    containsair, containsgravel : Boolean;
begin
    index := 1;

    if (totaloutputs = 264)or(totaloutputs = 88) then
    begin
        for col := 0 to 9 do
        begin
            for row := 0 to 9 do
            begin
                topoutputpred[col,row] := outpipe;
                bottomoutputpred[col,row] := outpipe;
            end;
        end;
        for col := 0 to 9 do
        begin
            case col of
                0: begin top := 2; bottom := 7; end;
                1: begin top := 1; bottom := 8; end;
                2..7: begin top := 0; bottom := 9; end;
                8: begin top := 1; bottom := 8; end;
                9: begin top := 2; bottom := 7; end;
            end;
            for row := top to bottom do
            begin
                if totaloutputs = 264 then
                begin
                    if (outputstore[testposition].prediction[index+2]
                        >=outputstore[testposition].prediction[index+1])
                    and(outputstore[testposition].prediction[index+2]
                        >=outputstore[testposition].prediction[index])then
                        topphasepred := air
                    else if (outputstore[testposition].prediction[index]
                        >=outputstore[testposition].prediction[index+1])
                    and(outputstore[testposition].prediction[index]
                        >outputstore[testposition].prediction[index+2])then
                        topphasepred := water
                    else
                        topphasepred := gravel;
                    end
                else
                begin
                    RecLoadDBase.GetPhases(containsair,containsgravel);
                    if outputstore[testposition].prediction[index] >= 0 then
                    begin
                        if containsair then
                            topphasepred := air
                        else

```

```

topphasepred := gravel;
                    end
                else
                    topphasepred := water;
                end;
            end;
        end;
        topoutputpred[col,row] := topphasepred;
        if topphasepred = air then
            volumestore[testposition,1]:=volumestore[testposition,1]+1
        else if topphasepred = gravel then
            volumestore[testposition,2]:=volumestore[testposition,2]+1;

        if totaloutputs = 264 then
        begin
            if (outputstore[testposition].prediction[index+2+264]
                >=outputstore[testposition].prediction[index+1+264])
            and(outputstore[testposition].prediction[index+2+264]
                >=outputstore[testposition].prediction[index+264])then
                bottomphasepred := air
            else if (outputstore[testposition].prediction[index+264]
                >=outputstore[testposition].prediction[index+1+264])
            and(outputstore[testposition].prediction[index+264]
                >outputstore[testposition].prediction[index+2+264])then
                bottomphasepred := water
            else
                bottomphasepred := gravel;

            inc(index,3);
        end
        else
        begin
            if outputstore[testposition].prediction[index+88] >= 0 then
            begin
                if containsair then
                    bottomphasepred := air
                else
                    bottomphasepred := gravel;
            end
            else
                bottomphasepred := water;

            inc(index,1);
        end;

        bottomoutputpred[col,row] := bottomphasepred;
        if bottomphasepred = air then
            volumestore[testposition,3]:=volumestore[testposition,3]+1
        else if bottomphasepred = gravel then
            volumestore[testposition,4]:=volumestore[testposition,4]+1;

        end;

    end;

    if volumestore[testposition,1] < topairthreshold then
        volumestore[testposition,1] := 0;
    if volumestore[testposition,2] < topgravthreshold then
        volumestore[testposition,2] := 0;
    if volumestore[testposition,3] < botairthreshold then
        volumestore[testposition,3] := 0;
    if volumestore[testposition,4] < botgravthreshold then
        volumestore[testposition,4] := 0;

    if (testposition mod 5 = 0) then
    begin
        TopTomo.Desired := topoutputpred;
        BottomTomo.Desired := bottomoutputpred;
        TopTomo.Repaint;
        BottomTomo.Repaint;
        frmnumber.Caption := inttostr(recomposition);
        DecodeTime(outputstore[recomposition].sampletime,hours,mins,secs,msecs);
        smins.Caption := inttostr(mins)+' min';
        ssecs.Caption := inttostr(secs)+' s';
        smsecs.Caption := inttostr(msecs)+' ms';
        Update;
    end;
end
else
begin
    if (testposition mod 5 = 0) then
    begin
        TopAirChart.Series[0].Clear;

```

```

TopAirChart.Series[0].Addy(outputstore[testposition].prediction[1],
    ",cTeeColor);
TopGravChart.Series[0].Clear;
TopGravChart.Series[0].Addy(outputstore[testposition].prediction[2],
    ",cTeeColor);
TopWaterChart.Series[0].Clear;
TopWaterChart.Series[0].Addy(100-(outputstore[testposition].prediction[1]+
    outputstore[testposition].prediction[2]),",cTeeColor);
BottomAirChart.Series[0].Clear;
BottomAirChart.Series[0].Addy(outputstore[testposition].prediction[3],
    ",cTeeColor);
BottomGravChart.Series[0].Clear;
BottomGravChart.Series[0].Addy(outputstore[testposition].prediction[4],
    ",cTeeColor);
BottomWaterChart.Series[0].Clear;
BottomWaterChart.Series[0].Addy(100-(outputstore[testposition].prediction[3]+
    outputstore[testposition].prediction[4]),",cTeeColor);
frmnumber.Caption := inttostr(recomposition);
DecodeTime(outputstore[recomposition].sampletime, hours, mins, secs, msecs);
smmins.Caption := inttostr(mins)+' min';
sssecs.Caption := inttostr(secs)+' s';
smsecs.Caption := inttostr(msecs)+' ms';
Update;
end;
end;
end;
end;

```

(* perform a point-by-point cross-correlation on the volume fraction data obtained from top and bottom measurement systems - the peak of this function gives the number of frames difference between the top and bottom systems. By multiplying this number of frames by the frame interval, the transit time of the gravel and sir bubbles is obtained. Since the system separation is known, this is then used to calculate the average velocity between the top and bottom system *)

```

procedure TRealMain.PerformCorrelation;
var testindex : integer;
    upperindex : integer;
    i,k, maxindex : integer;
    tempsum, maxval : real;
    denominator : real;
begin
    // perform direct correlation on the air volume fraction data from
    // the top and bottom system
    upperindex := DesiredFrames.Value;
    maxval := 0;
    for i := 0 to upperindex do
        begin
            tempsum := 0;
            for k := 1 to upperindex do
                begin
                    if (k+i) <= upperindex then
                        begin
                            if totaloutputs = 2 then
                                begin
                                    if StepperConfig.FlowDirSelect.ItemIndex = 1 then
                                        tempsum := tempsum + outputstore[k+i].prediction[1]*
                                            outputstore[k].prediction[3]
                                    else
                                        tempsum := tempsum + outputstore[k+i].prediction[3]*
                                            outputstore[k].prediction[1];
                                    end
                                end
                            else
                                begin
                                    if StepperConfig.FlowDirSelect.ItemIndex = 1 then
                                        tempsum := tempsum + volumestore[k+i,1]*
                                            volumestore[k,3]
                                    else
                                        tempsum := tempsum + volumestore[k+i,3]*
                                            volumestore[k,1];
                                    end
                                end
                            end;
                        end;
                    end;
                end;
            aircorr[i+1] := tempsum/upperindex;
            // keep a record of the maximum of the correlation function as well
            // as the corresponding frame number
            if aircorr[i+1] > maxval then
                begin
                    maxval := aircorr[i+1];
                    maxindex := i+1;
                end;
            end;
        end;
    end;
end;

```

```

// calculate the transit time for the air bubble
airdelay := maxindex-1;
denominator := airdelay*frameinterval;
if (denominator = 0) or (maxval < correlationthreshold) then
    begin
        airvelocity := -1;
        AirVelPredict.Caption := 'unknown'
    end
else
    begin
        // if non-zero then calculate the average velocity of the air bubble
        airvelocity := (flowmeterfactor*systemseparation)/denominator;
        if airvelocity < 0.001 then
            airvelocity := 0;
        AirVelPredict.Caption := floattostrf(airvelocity,ffFixed,5,3);
    end;
end;

```

```

// repeat the process for the gravel bubble
maxval := 0;
for i := 0 to upperindex do
    begin
        tempsum := 0;
        for k := 1 to upperindex do
            begin
                if (k+i)<=upperindex then
                    begin
                        if totaloutputs = 2 then
                            begin
                                if StepperConfig.FlowDirSelect.ItemIndex = 1 then
                                    tempsum := tempsum + outputstore[k+i].prediction[2]*
                                        outputstore[k].prediction[4]
                                else
                                    tempsum := tempsum + outputstore[k+i].prediction[4]*
                                        outputstore[k].prediction[2];
                                end
                            end
                        else
                            begin
                                if StepperConfig.FlowDirSelect.ItemIndex = 1 then
                                    tempsum := tempsum + volumestore[k+i,2]*
                                        volumestore[k,4]
                                else
                                    tempsum := tempsum + volumestore[k+i,4]*
                                        volumestore[k,2];
                                end
                            end;
                        end;
                    end;
                end;
            gravelcorr[i+1] := tempsum/upperindex;
            if gravelcorr[i+1] > maxval then
                begin
                    maxval := gravelcorr[i+1];
                    maxindex := i+1;
                end;
            end;
        end;
        graveldelay := maxindex-1;
        denominator := graveldelay*frameinterval;
        if (denominator = 0) or (maxval < correlationthreshold) then
            begin
                gravelvelocity := -1;
                GravelVelPredict.Caption := 'unknown';
            end
        else
            begin
                gravelvelocity := (flowmeterfactor*systemseparation)/denominator;
                if gravelvelocity < 0.001 then
                    gravelvelocity := 0;
                GravelVelPredict.Caption := floattostrf(gravelvelocity,ffFixed,5,3);
            end;
        end;
    end;
end;

```

```

//plot results
for testindex := 1 to upperindex do
    begin
        if totaloutputs = 2 then
            begin
                AirPlot.Series[0].AddY(outputstore[testindex].prediction[1]);
                AirPlot.Series[1].AddY(outputstore[testindex].prediction[3]);
                GravelPlot.Series[0].AddY(outputstore[testindex].prediction[2]);
                GravelPlot.Series[1].AddY(outputstore[testindex].prediction[4]);
            end
        else
            begin
                AirPlot.Series[0].AddY(volumestore[testindex,1]);
            end
        end;
    end;
end;

```

```

    AirPlot.Series[1].AddY(volumestore[testindex,3]);
    GravelPlot.Series[0].AddY(volumestore[testindex,2]);
    GravelPlot.Series[1].AddY(volumestore[testindex,4]);
end;
AirCorrPlot.Series[0].AddY(aircorr[testindex]);
GravelCorrPlot.Series[0].AddY(gravellcorr[testindex]);
end;
end;

(* ----- STEPPER CONFIGURATION PROCEDURES ----- *)
(* configure the stepper motors for a test *)
procedure TRealMain.MotorSetupClick(Sender: TObject);
begin
    StepperInit;
    StepperConfig.ShowModal;
end;

(* stop stepper drives immediately *)
procedure TRealMain.StepperStopClick(Sender: TObject);
begin
    Stepper.StopDrive;
end;

(* start stepper drives - based on the current test configuration, calculate
the direction, speed and distance for a fixed distance move *)
procedure TRealMain.RealStartMotorsClick(Sender: TObject);
var tempsteps : integer;
begin
    StepperStop.Visible := true;
    StepperInit;
    if StepperConfig.airrps > 0 then
        begin
            if StepperConfig.FlowDirSelect.ItemIndex = 0 then
                begin
                    if StepperConfig.AirBubblePosition.ItemIndex = 0 then
                        tempsteps := StepperConfig.airsteps
                    else
                        tempsteps := -StepperConfig.airsteps;
                    end
                end
            else
                begin
                    if StepperConfig.AirBubblePosition.ItemIndex = 0 then
                        tempsteps := -StepperConfig.airsteps
                    else
                        tempsteps := StepperConfig.airsteps;
                    end
                end
            end;

            case StepperConfig.AirDrive.Value of
                1: begin
                    drive1rps := StepperConfig.airrps;
                    drive1steps := tempsteps;
                    end;
                2: begin
                    drive2rps := StepperConfig.airrps;
                    drive2steps := -tempsteps;
                    end;
                3: begin
                    drive3rps := StepperConfig.airrps;
                    drive3steps := tempsteps;
                    end;
            end;
        end;
    end;

    if StepperConfig.gravelrps > 0 then
        begin
            if StepperConfig.FlowDirSelect.ItemIndex = 0 then
                begin
                    if StepperConfig.GravelBubblePosition.ItemIndex = 0 then
                        tempsteps := StepperConfig.gravelsteps
                    else
                        tempsteps := -StepperConfig.gravelsteps;
                    end
                end
            else
                begin
                    if StepperConfig.GravelBubblePosition.ItemIndex = 0 then
                        tempsteps := -StepperConfig.gravelsteps
                    else
                        tempsteps := StepperConfig.gravelsteps;
                    end
                end
            end;

            case StepperConfig.GravelDrive.Value of

```

```

                1: begin
                    drive1rps := StepperConfig.gravelrps;
                    drive1steps := tempsteps;
                    end;
                2: begin
                    drive2rps := StepperConfig.gravelrps;
                    drive2steps := -tempsteps;
                    end;
                3: begin
                    drive3rps := StepperConfig.gravelrps;
                    drive3steps := tempsteps;
                    end;
            end;
        end;
    end;
    Stepper.MoveFixedDist(drive1rps,drive2rps,drive3rps,drive1steps,
        drive2steps,drive3steps);
end;

(* initialise the stepper motor variables *)
procedure TRealMain.StepperInit;
begin
    drive1rps := 0;
    drive2rps := 0;
    drive3rps := 0;
    drive1steps := 0;
    drive2steps := 0;
    drive3steps := 0;
end;

(* ----- RESULT VERIFICATION PROCEDURES ----- *)
(* to verify the dynamic performane of a test, show the dynverify form *)
procedure TRealMain.VerifyResultsClick(Sender: TObject);
begin
    DynVerify.ShowModal;
end;

(* to view real-time results after a test, enable the realtimer *)
procedure TRealMain.RealViewClick(Sender: TObject);
begin
    if RealTimer.Enabled then
        begin
            RealTimer.Enabled := false;
            RealView.Caption := 'Continue';
        end
    else
        begin
            RealTimer.Enabled := true;
            RealView.Caption := 'Stop tests';
        end
    end;
end;

(* each time the timer overflows, show the network prediction for that particular
frame, show the sample time of that particular frame and move viewposition
to the next frame in the matrix of results *)
procedure TRealMain.RealTimerTimer(Sender: TObject);
begin
    RealTimer.Interval := floor(1000*(1/RealFrameRate.Value));
    PlotTestOutput(viewposition);
    frmnumber.Caption := inttostr(viewposition);
    DecodeTime(outputstore[viewposition].sampletime,hours,mins,secs,msecs);
    smins.Caption := inttostr(mins)+' min';
    ssecs.Caption := inttostr(secs)+' s';
    smsecs.Caption := inttostr(msecs)+' ms';
    inc(viewposition,ViewInc.Value);
    // if have reached the end of the results, then loop back to the start
    if viewposition > DesiredFrames.Value then
        viewposition := ViewInc.Value;
    end;
end;

(* ----- FILE HANDLING PROCEDURES ----- *)
(* load the neural network weights from the specified files *)
procedure TRealMain.RealWeightsClick(Sender: TObject);
var tempfile : textfile;
    filename : string;
    tempair, tempgrav : Boolean;
    index, temphidden, tempoutputs, colindex, rowindex, tempphase : integer;
begin
    upperindex := 128;
    numtophidden := 0;
    numbottomhidden := 0;
    tempair := false;

```

```

tempgrav := false;

filename := 'c:\TestData\weight'+inttostr(TopTestNum.Value)+'.txt';
AssignFile(tempfile,filename);
Reset(tempfile);
ReadLn(tempfile,tempoutputs);
totaloutputs := tempoutputs;
// load the number of outputs from the files and set the neural
// network output display appropriately
if totaloutputs = 2 then
begin
    TopVolume.Visible := true;
    BottomVolume.Visible := true;
end
else
begin
    TopVolume.Visible := false;
    BottomVolume.Visible := false;
end;

// load the weights for the top system
ReadLn(tempfile,temphidden);
if temphidden > 0 then
begin
    upperindex := temphidden;
    numtophidden := temphidden;
    SetLength(tophiddenweights,129,temphidden);
    SetLength(tophiddenoutputs,temphidden+1);
    // load in the weights for the hidden layer neurons
    for colindex := 0 to temphidden - 1 do
        for rowindex := 0 to 128 do
            ReadLn(tempfile,tophiddenweights[rowindex,colindex]);
        end;
    end;

    SetLength(topweights,upperindex+1,totaloutputs);
    // load in the weights for the output layer neurons
    for colindex := 0 to tempoutputs - 1 do
        for rowindex := 0 to upperindex do
            ReadLn(tempfile,topweights[rowindex,colindex]);
        end;
    end;

    CloseFile(tempfile);

    // load the weights for the bottom system
    upperindex := 128;
    filename := 'c:\TestData\weight'+inttostr(BottomTestNum.Value)+'.txt';
    AssignFile(tempfile,filename);
    Reset(tempfile);
    ReadLn(tempfile,tempoutputs);
    ReadLn(tempfile,temphidden);
    if temphidden > 0 then
    begin
        upperindex := temphidden;
        numbottomhidden := temphidden;
        SetLength(bottomhiddenweights,129,temphidden);
        SetLength(bottomhiddenoutputs,temphidden+1);
        // load in the weights for the hidden layer neurons
        for colindex := 0 to temphidden - 1 do
            for rowindex := 0 to 128 do
                ReadLn(tempfile,bottomhiddenweights[rowindex,colindex]);
            end;
        end;

        SetLength(bottomweights,upperindex+1,totaloutputs);
        // load in the weights for the output layer neurons
        for colindex := 0 to tempoutputs - 1 do
            for rowindex := 0 to upperindex do
                ReadLn(tempfile,bottomweights[rowindex,colindex]);
            end;
        end;

        CloseFile(tempfile);

        // load the pre-processing data for the top system
        filename := 'c:\TestData\preproc'+inttostr(TopTestNum.Value)+'.txt';
        AssignFile(tempfile,filename);
        Reset(tempfile);
        for index := 1 to 128 do
            ReadLn(tempfile,topmean[index]);
        end;
        for index := 1 to 128 do
            ReadLn(tempfile,topstddev[index]);
        end;
        if totaloutputs = 88 then
        begin
            ReadLn(tempfile,tempphase);
            if tempphase = 0 then
                tempair := true;
            else
                tempgrav := true;
            RecLoadDBase.SetPhases(tempair,tempgrav);
        end;
        CloseFile(tempfile);

        // load the pre-processing data for the bottom system
        filename := 'c:\TestData\preproc'+inttostr(BottomTestNum.Value)+'.txt';
        AssignFile(tempfile,filename);
        Reset(tempfile);
        for index := 1 to 128 do
            ReadLn(tempfile,bottommean[index]);
        end;
        for index := 1 to 128 do
            ReadLn(tempfile,bottomstddev[index]);
        end;
        CloseFile(tempfile);
    end;

    (* ----- GENERAL PROCEDURES ----- *)
    (* when exit, deallocate the dynamic memory and stepper motor resources and
    save the settings to the registry *)
    procedure TRealMain.RealExitClick(Sender: TObject);
    var Reg: TRegistry;
    begin
        Reg := TRegistry.Create;
        try
            Reg.OpenKey('Software\Neural', True);
            Reg.WriteInteger('RealOnline',RealOnline.ItemIndex);
            Reg.WriteInteger('TopTestNumber',TopTestNum.Value);
            Reg.WriteInteger('BottomTestNumber',BottomTestNum.Value);
            Reg.WriteInteger('RealFrameRate',RealFrameRate.Value);
            Reg.WriteInteger('DesiredFrames',DesiredFrames.Value);
            Reg.WriteString('FlowFactor',FlowFactorInput.Text);
            Reg.WriteString('Separation',SystemSepInput.Text);
            Reg.WriteString('TopAirThresh',TopAirThresh.Text);
            Reg.WriteString('TopGravThresh',TopGravThresh.Text);
            Reg.WriteString('BotAirThresh',BotAirThresh.Text);
            Reg.WriteString('BotGravThresh',BotGravThresh.Text);
            Reg.WriteString('CorrThresh',CorrThreshInp.Text);
            Reg.WriteInteger('ViewInc',ViewInc.Value);
        finally
            Reg.Free;
        end;
        topweights := nil;
        tophiddenweights := nil;
        tophiddenoutputs := nil;
        bottomhiddenoutputs := nil;
        bottomweights := nil;
        bottomhiddenweights := nil;
        Stepper.Free;
        Close;
    end;

    (* when open form, load previous settings from the registry *)
    procedure TRealMain.FormShow(Sender: TObject);
    var Reg: TRegistry;
    KeyGood: Boolean;
    begin
        Stepper := TStep.Create;
        Reg := TRegistry.Create;
        try
            KeyGood := Reg.OpenKey('Software\Neural', False);
            if KeyGood then
                RealOnline.ItemIndex := Reg.ReadInteger('RealOnline');
                TopTestNum.Value := Reg.ReadInteger('TopTestNumber');
                BottomTestNum.Value := Reg.ReadInteger('BottomTestNumber');
                RealFrameRate.Value := Reg.ReadInteger('RealFrameRate');
                DesiredFrames.Value := Reg.ReadInteger('DesiredFrames');
                SystemSepInput.Text := Reg.ReadString('Separation');
                FlowFactorInput.Text := Reg.ReadString('FlowFactor');
                TopAirThresh.Text := Reg.ReadString('TopAirThresh');
                TopGravThresh.Text := Reg.ReadString('TopGravThresh');
                BotAirThresh.Text := Reg.ReadString('BotAirThresh');
                BotGravThresh.Text := Reg.ReadString('BotGravThresh');
                CorrThreshInp.Text := Reg.ReadString('CorrThresh');
                ViewInc.Value := Reg.ReadInteger('ViewInc');
                DynVerify.AirDiameter.Text := Reg.ReadString('AirDiameter');
                DynVerify.AirLength.Text := Reg.ReadString('AirLength');
                DynVerify.AirBubbleStartPosition.Text :=
                    Reg.ReadString('AirBubbleStartPosition');
            end;
        finally
            Reg.Free;
        end;
    end;

```

```

        DynVerify.GravelDiameter.Text := Reg.ReadString('GravelDiameter');
        DynVerify.GravelLength.Text := Reg.ReadString('GravelLength');
        DynVerify.GravelBubbleStartPosition.Text :=
Reg.ReadString('GravelBubbleStartPosition');
        DynVerify.airposition := Reg.ReadInteger('AirPosition');
        DynVerify.gravelposition := Reg.ReadInteger('GravelPosition');
        DynVerify.UpperSystemTop.Text := Reg.ReadString('UpperSystTop');
        DynVerify.UpperSystemBottom.Text := Reg.ReadString('UpperSystBot');
        DynVerify.LowerSystemTop.Text := Reg.ReadString('LowerSystTop');
        DynVerify.LowerSystemBottom.Text := Reg.ReadString('LowerSystBot');
        DynVerify.VelSource.ItemIndex := Reg.ReadInteger('VelSource');
    finally
        Reg.Free;
    end;

    VerifyResults.Visible := false;
    StepperStop.Visible := false;
    viewposition := ViewInc.Value;
    notifvalue := SampHardware.InitSample(1,Handle);
    curcalibrate := false;
end;

(* to perform a calibration, initiate a capture of 50 frames of data for the
rig filled only with water *)
procedure TRealMain.RealCalibrateClick(Sender: TObject);
begin
    curcalibrate := true;
    SampHardware.StartSample(50);
end;

(* override the default Windows message handling procedure to capture the
sampling complete message - when sampling is complete call StopSampling and
process the results *)
procedure TRealMain.WndProc(var TheMsg:TMessage);
begin
    if TheMsg.Msg = notifvalue then
    begin
        SampHardware.StopSample(result);
        ProcessResults;
    end
    else
        inherited WndProc(TheMsg);
end;

end.

```

PROGRAM LISTING OF STEPCONFIG.PAS

(* This unit is used to set up the stepper motors for a dynamic test. Jog features allow the user to carefully position the bubble at one of the defined starting positions before a test. To calculate how many steps are required to move the bubble a certain distance at a certain speed, the diameter of the pulley must be specified. The user stipulates whether an upward or downward flow is desired and whether the bubble is on the inside or the outside of the pulley relative to the stand, and this unit calculates in which direction the motor must step. If the AT6400 stepper controller has been switched off since the previous test then it is necessary to download the operating system, otherwise the stepper motors simply need to be re-initialised. *)

```

unit stepconfig;

interface

uses
    Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
    Buttons, ExtCtrls, StdCtrls, Registry, StepUnit, Spin, Math;

type
    TStepperConfig = class(TForm)
    Panel1: TPanel;
    Label1: TLabel;
    Panel2: TPanel;
    StepClose: TSpeedButton;
    Label2: TLabel;
    Bevel1: TBevel;
    StepDownload: TSpeedButton;
    StepInitialise: TSpeedButton;
    end;

```

```

StepReset: TSpeedButton;
Label3: TLabel;
Label4: TLabel;
Label5: TLabel;
Label6: TLabel;
Label7: TLabel;
AirDiameter: TEdit;
AirVelocity: TEdit;
AirDistance: TEdit;
Label9: TLabel;
GravelDiameter: TEdit;
GravelVelocity: TEdit;
Label10: TLabel;
Label11: TLabel;
GravelDistance: TEdit;
AirBubblePosition: TRadioGroup;
GravelBubblePosition: TRadioGroup;
FlowDirSelect: TRadioGroup;
AirDrive: TSpinEdit;
GravelDrive: TSpinEdit;
Label13: TLabel;
Label14: TLabel;
AirAccel: TEdit;
Label8: TLabel;
Label12: TLabel;
GravelAccel: TEdit;
AirBubbleJog: TPanel;
GravelBubbleJog: TPanel;
Label15: TLabel;
AddPhaseSelect: TComboBox;
procedure StepCloseClick(Sender: TObject);
procedure FormShow(Sender: TObject);
procedure StepDownloadClick(Sender: TObject);
procedure StepInitialiseClick(Sender: TObject);
procedure StepResetClick(Sender: TObject);
procedure AirBubbleJogMouseDown(Sender: TObject; Button: TMouseButton;
    Shift: TShiftState; X, Y: Integer);
procedure AirBubbleJogMouseUp(Sender: TObject; Button: TMouseButton;
    Shift: TShiftState; X, Y: Integer);
procedure GravelBubbleJogMouseDown(Sender: TObject;
    Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
procedure GravelBubbleJogMouseUp(Sender: TObject; Button: TMouseButton;
    Shift: TShiftState; X, Y: Integer);
end;

```

```

private
{ Private declarations }
public
{ Public declarations }
airsteps : integer;           // number of steps of air drive
airrps : real;                // revolutions per second of air drive
airrevs : real;               // revolutions of air drive
gravelsteps : integer;        // number of steps of gravel drive
gravelrps : real;             // revolutions per second of gravel drive
gravelrevs : real;            // revolutions of gravel drive
aircircumference : real;      // air pulley circumference
gravelcircumference : real;   // gravel pulley circumference
airacceleration : integer;
gravacceleration : integer;
end;

```

```

var
    StepperConfig: TStepperConfig;
implementation

```

```

uses realunit,dynunit;

```

```

{$R *.DFM}

```

```

(* ----- GENERAL PROCEDURES ----- *)
(* when close form, calculate what the required steps and steps/second for
each of the drives based on the specified test parameters *)
procedure TStepperConfig.StepCloseClick(Sender: TObject);
var Reg: TRegistry;
tempaccel1, tempaccel2, tempaccel3 : integer;
begin
    // write the current settings to the registry
    Reg := TRegistry.Create;
    try
        Reg.OpenKey('Software\Neural', True);
        Reg.WriteString('AirPulley',AirDiameter.Text);
        Reg.WriteString('GravelPulley',GravelDiameter.Text);
        Reg.WriteString('AirVelocity',AirVelocity.Text);
    end;
end;

```



```

Reg.WriteString('GravelVelocity',GravelVelocity.Text);
Reg.WriteString('AirDistance',AirDistance.Text);
Reg.WriteString('GravelDistance',GravelDistance.Text);
Reg.WriteLine('AirBubblePosition',AirBubblePosition.ItemIndex);
Reg.WriteLine('GravelBubblePosition',GravelBubblePosition.ItemIndex);
Reg.WriteLine('FlowDirection',FlowDirSelect.ItemIndex);
Reg.WriteLine('AirDrive',AirDrive.Value);
Reg.WriteLine('GravelDrive',GravelDrive.Value);
Reg.WriteString('AirAccel',AirAccel.Text);
Reg.WriteString('GravelAccel',GravelAccel.Text);
finally
  Reg.Free;
end;
// calculate the number of revolutions for each drive based on
// the desired distance and the circumference of the pulley
aircircumference := pi*strtofloat(AirDiameter.Text);
gravircumference := pi*strtofloat(GravelDiameter.Text);
airrps := strtofloat(AirVelocity.Text)/aircircumference;
gravlrps := strtofloat(GravelVelocity.Text)/gravircumference;
// assuming 400 steps per revolution drive resolution, convert
// the number of revolutions into the number of steps
airrevs := strtofloat(AirDistance.Text)/aircircumference;
gravirevs := strtofloat(GravelDistance.Text)/gravircumference;
airsteps := round(400*airrevs);
gravelsteps := round(400*gravirevs);

tempaccel1 := 50;
tempaccel2 := 50;
tempaccel3 := 50;
airacceleration := strtoint(AirAccel.Text);
gravacceleration := strtoint(GravelAccel.Text);

case AirDrive.Value of
  1: begin tempaccel1 := airacceleration; end;
  2: begin tempaccel2 := airacceleration; end;
  3: begin tempaccel3 := airacceleration; end;
end;
case GravelDrive.Value of
  1: begin tempaccel1 := gravacceleration; end;
  2: begin tempaccel2 := gravacceleration; end;
  3: begin tempaccel3 := gravacceleration; end;
end;
RealMain.Stepper.SetAcceleration(tempaccel1,tempaccel2,tempaccel3);
Close;
end;

(* when show form, load the previous settings from the registry *)
procedure TStepperConfig.FormShow(Sender: TObject);
var Reg : TRegistry;
    KeyGood: Boolean;
begin
  Reg := TRegistry.Create;
  try
    KeyGood := Reg.OpenKey('Software\Neural', False);
    if KeyGood then
      AirDiameter.Text := Reg.ReadString('AirPulley');
      GravelDiameter.Text := Reg.ReadString('GravelPulley');
      AirVelocity.Text := Reg.ReadString('AirVelocity');
      GravelVelocity.Text := Reg.ReadString('GravelVelocity');
      AirDistance.Text := Reg.ReadString('AirDistance');
      GravelDistance.Text := Reg.ReadString('GravelDistance');
      AirBubblePosition.ItemIndex := Reg.ReadInteger('AirBubblePosition');
      GravelBubblePosition.ItemIndex := Reg.ReadInteger('GravelBubblePosition');
      FlowDirSelect.ItemIndex := Reg.ReadInteger('FlowDirection');
      AirDrive.Value := Reg.ReadInteger('AirDrive');
      GravelDrive.Value := Reg.ReadInteger('GravelDrive');
      AirAccel.Text := Reg.ReadString('AirAccel');
      GravelAccel.Text := Reg.ReadString('GravelAccel');
    finally
      Reg.Free;
    end;
  end;
end;

(* ----- INITIALISATION PROCEDURES ----- *)
(* download the AT6400 operating system *)
procedure TStepperConfig.StepDownloadClick(Sender: TObject);
begin
  RealMain.Stepper.DownloadOS;
end;

(* initialise the drives using the standard settings found in 'init.txt'

```

```

textfile that must be located in the same directory as this program *)
procedure TStepperConfig.StepInitialiseClick(Sender: TObject);
begin
  RealMain.Stepper.Initialise(768,'init.txt');
  RealMain.Stepper.SetZeroPos;
end;

(* reset the stepper drive positions to zero *)
procedure TStepperConfig.StepResetClick(Sender: TObject);
begin
  RealMain.Stepper.ResetLimits;
  RealMain.Stepper.ResetDrives;
end;

(* ----- BUBBLE MOVEMENT PROCEDURES ----- *)
(* jog the air bubble - if left click then bubble moves upwards and if you
right click then the bubble moves upwards. NOTE: the bubble continues to move
while button is pressed and, since no hardware limits set, user must know where
bubble is positioned before starting a move *)
procedure TStepperConfig.AirBubbleJogMouseDown(Sender: TObject;
  Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
var tempdir: TDirection;
begin
  (* upward movement *)
  if ssLeft in Shift then
    begin
      if (AirDrive.Value = 1)or(AirDrive.Value = 3)then
        begin
          if AirBubblePosition.ItemIndex = 1 then
            tempdir := CW
          else
            tempdir := CCW;
          end
        end
      else
        begin
          if AirBubblePosition.ItemIndex = 1 then
            tempdir := CCW
          else
            tempdir := CW;
          end;
        end
      RealMain.Stepper.JogMove(AirDrive.Value,tempdir);
    end
  (* downward movement *)
  else if ssRight in Shift then
    begin
      if (AirDrive.Value = 1)or(AirDrive.Value = 3) then
        begin
          if AirBubblePosition.ItemIndex = 1 then
            tempdir := CCW
          else
            tempdir := CW;
          end
        end
      else
        begin
          if AirBubblePosition.ItemIndex = 1 then
            tempdir := CW
          else
            tempdir := CCW;
          end;
        end
      RealMain.Stepper.JogMove(AirDrive.Value,tempdir);
    end;
  end;

(* when the mouse button is released, stop the bubble movement *)
procedure TStepperConfig.AirBubbleJogMouseUp(Sender: TObject;
  Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
begin
  RealMain.Stepper.StopDrive;
end;

(* jog the gravel bubble - if left click then bubble moves upward else if
right click then bubble moves downwards *)
procedure TStepperConfig.GravelBubbleJogMouseDown(Sender: TObject;
  Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
var tempdir : TDirection;
begin
  (* upward movement *)
  if ssLeft in Shift then
    begin
      if (GravelDrive.Value = 1)or(GravelDrive.Value = 3) then
        begin

```

```

        if GravelBubblePosition.ItemIndex = 1 then
            tempdir := CW
        else
            tempdir := CCW;
        end
    else
        begin
            if GravelBubblePosition.ItemIndex = 1 then
                tempdir := CCW
            else
                tempdir := CW;
            end;
            RealMain.Stepper.JogMove(GravelDrive.Value,tempdir);
        end
        (* downward movement *)
    else if ssRight in Shift then
        begin
            if (GravelDrive.Value = 1)or(GravelDrive.Value = 3) then
                begin
                    if GravelBubblePosition.ItemIndex = 1 then
                        tempdir := CCW
                    else
                        tempdir := CW;
                    end
                end
            else
                begin
                    if GravelBubblePosition.ItemIndex = 1 then
                        tempdir := CW
                    else
                        tempdir := CCW;
                    end;
                    RealMain.Stepper.JogMove(GravelDrive.Value,tempdir);
                end;
            end;
        end;

    (* once the mouse button is released, stop the bubble movement *)
    procedure TStepperConfig.GravelBubbleJogMouseUp(Sender: TObject;
        Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
    begin
        RealMain.Stepper.StopDrive;
    end;

end.

```

PROGRAM LISTING OF STEPUNIT.PAS

(* This unit it used to send and receive commands from the AT6400 4-axis controller and in so doing, control the motion of the two stepper motors connected for generating test cases. The SendCommand and ReceiveCommand procedures were ported from a Pascal device driver provided by Compumotor. Since Delphi does not support the PORT and PORTW commands directly, a special component called 'simport' was installed to perform this task. To download the AT6400 operating system ,the AT6400.EXE program provided by Compumotor is executed. *)

```
unit StepUnit;
```

```
interface
```

```
uses sysutils, simport, windows;
```

```
const
```

```
// AT6400 masks
```

```
OB_HAS_DATA   = $01; // AT6400 output buffer has data
IB_IS_EMPTY   = $02; // AT6400 input buffer is empty
GP_INTERRUPT   = $04; // AT6400 general purpose interrupt
CS_UPDATED    = $08; // AT6400 card status update
MASTER        = $10; // AT6400 master interrupt enable
KILL_REQ      = $20; // AT6400 kill has been requested
OS_LOADED     = $40; // AT6400 operating system loaded
XI_LOADED     = $80; // AT6400 Xilinx part loaded
```

```
// AT6400 commands
```

```
CMD_READY     = $42; // tells AT6400 that input buffer has data
CLEAR_GPINT   = $44; // tells AT6400 to clear general purpose interrupt
REQ_STATUS    = $48; // tells AT6400 that you want a status update
REQ_KILL      = $81; // tells AT6400 that you want the kill command executed
```

```
// AT6400 address offsets
```

```
FASTSTATUS    = 2; // fast status offset
STATUS        = 4; // set/clear status offset
```

```
// AT6400 fast status
```

```

    AXIS1_MOTOR   = $00;
    AXIS2_MOTOR   = $02;
    AXIS3_MOTOR   = $04;
    AXIS4_MOTOR   = $06;
    AXIS1_ENCODER = $08;
    AXIS2_ENCODER = $0A;
    AXIS3_ENCODER = $0C;
    AXIS4_ENCODER = $0E;
    AXIS1_VELOCITY = $10;
    AXIS2_VELOCITY = $12;
    AXIS3_VELOCITY = $14;
    AXIS4_VELOCITY = $16;
    AXIS1_STATUS  = $18;
    AXIS2_STATUS  = $1A;
    AXIS3_STATUS  = $1C;
    AXIS4_STATUS  = $1E;
    INPUT_STATUS  = $20;
    OUTPUT_STATUS = $22;
    LIMIT_STATUS  = $24;
    INO_STATUS    = $25;
    ANALOG_STATUS = $26;
    INT_STATUS    = $28;
    SYSTEM_STATUS = $2A;
    USER_STATUS   = $2C;
    TFRM_STATUS   = $2D;
    TIMER_STATUS  = $2E;

```

```
// Maximums
```

```

    MAXBYTES = 256;
    MAXWORDS = 128;

```

```
// Genral purpose constants
```

```

    MAXLINE   = 76;
    BASEP     = $0300;
    BACKSPACE = #8;
    ENTER      = #13;
    LINEFEED   = #10;
    SPACE      = #32;
    ESCAPE     = #27;
    DELETE     = #126;
    CTRLK      = #11;
    TERMINATOR = #0;

```

```
type
```

```
CharBuffer = array[0..257] of char;
```

```
WordBuffer = array[0..128] of word;
```

```
TDirection = (CW,CCW); // specify motor direction
```

```
TRequestData = (MotorPos, Velocity); // specify what data requesting from // motor
```

```
TStep = class
```

```
private
```

```
    Address:word; // base address
```

```
    Command:CharBuffer; // command to AT6400
```

```
    Error:integer; // error integer code
```

```
    word_high: word; // used for status sampling
```

```
    word_low: word;
```

```
    fast_status: longint;
```

```
// send command to AT6400
```

```
function SendAT6400Block(Address:word; Command:CharBuffer):word;
```

```
// receive command from AT6400
```

```
function RecvAT6400Block(Address:word; var Response:CharBuffer):word;
```

```
// convert string command into appropriate data format
```

```
procedure ProcessCommand(temp: string);
```

```
// initiate an update status
```

```
procedure RequestStatus;
```

```
// get status from AT6400
```

```
procedure ReadStatus(var status_high, status_low:word; var status:longint);
```

```
// set what specific information want to retrieve
```

```
procedure SetPointer(status_offset:integer);
```

```
public
```

```
// load an initialisation file which contains a sequence of commands
```

```
// to initialise the stepper motors
```

```
procedure Initialise(NewAddress:word; readfilename: string);
```

```
// general purpose procedure which sets the velocity, and direction
```

```
// of both drives simultaneously
```

```
procedure GoDrive(drive1: Boolean; drive2: Boolean);
```

```

        drive3 : Boolean; vel1: integer; vel2: integer;
        vel3: integer; dir1: TDirection; dir2: TDirection;
        dir3: TDirection);
// move a drive continuously in a specific direction
procedure JogMove(drivenum: integer; dir: TDirection);
// move a drive a fixed distance at a certain velocity
procedure MoveFixedDist(vel1:real;vel2:real;vel3:real;dist1:integer;
        dist2:integer;dist3:integer);
// stop the motor drives
procedure StopDrive;
// disable the motor drives
procedure DisableDrives;
// reset the motor drives
procedure ResetDrives;
// get the status of the drives
procedure GetMotorData(want: TRequestData; var axis1: double;
        var axis2: double; var axis3: double);
// set the current drive position as the zero position for absolute
// movement
procedure SetZeroPos;
// download the operating system
procedure DownloadOS;
procedure ResetLimits;
procedure SetAcceleration(axis1:integer;axis2:integer;axis3:integer);
end;

implementation

(* ----- COMMAND/CONTROL PROCEDURES ----- *)
(* send a block of commands to the AT6400 *)
function TStep.SendAT6400Block(Address:word; Command:CharBuffer):word;
var
    l:word;
    Cmd:CharBuffer;
begin
    // clear out Cmd buffer
    for l:= 0 to MAXBYTES do
        begin
            Cmd[l] := TERMINATOR;
        end;
    // copy Command buffer to Cmd buffer
    l := 0;
    while( Command[l] <> TERMINATOR) do
        begin
            Cmd[l] := Command[l];
            Inc(l);
        end;
    // wait until is AT6400 is ready
    while( (Port[Address + STATUS] and IB_IS_EMPTY) <> 0 ) do
        begin
        end;
    // send command to AT6400
    l := 0;
    while( (WordBuffer(Cmd)[l] <> 0) and (l < MAXWORDS) ) do
        begin
            PortW[Address] := WordBuffer(Cmd)[l];
            Inc(l);
        end;
    // tell AT6400 we're done
    Port[Address + Status] := CMD_READY;
    // return the number of words successfully sent
    SendAT6400Block := l;
end;

(* receive a block of commands from the AT6400 *)
function TStep.RecvAT6400Block(Address:word; var Response:CharBuffer):word;
var
    l:word;
begin
    l := 0;
    while( ((Port[Address + STATUS] and OB_HAS_DATA) <> 0) and (l <
        MAXWORDS)) do
        begin
            WordBuffer(Response)[l] := PortW[Address];
            Inc(l);
        end;
    WordBuffer(Response)[l] := 0;
    Inc(l);
    // return the number of words received
    RecvAT6400Block := l;
end;

```

```

(* request the AT6400 to update its fast status area in memory *)
procedure TStep.RequestStatus;
begin
    port[address+STATUS] := REQ_STATUS; // request fast status update
    // wait for fast status information to be updated
    while((port[address+STATUS] and CS_UPDATED) = 0) do
        begin
        end;
    end;

(* read the status of the drives from the fast status area *)
procedure TStep.ReadStatus(var status_high, status_low:word; var status:longint);
begin
    status_high := portw[address+FASTSTATUS];
    status_low := portw[address+FASTSTATUS];
    // extract the upper and lower portions from the data
    status_high := (status_high shl 8) + (status_high shr 8);
    status_low := (status_low shl 8) + (status_low shr 8);
    status := status_high;
    status := (status shl 16) + status_low;
end;

(* set the fast status pointer to the status information that wish to access *)
procedure TStep.SetPointer(status_offset:integer);
begin
    port[address+FASTSTATUS] := status_offset;
end;

(* convert a string command into the appropriate format and send it to the AT6400 *)
procedure TStep.ProcessCommand(temp:string);
var CharPtr: byte;
begin
    for CharPtr := 0 to Length(temp)-1 do
        begin
            Command[CharPtr] := temp[CharPtr+1];
        end;
    Command[CharPtr] := ENTER;
    Inc(CharPtr);
    Command[CharPtr] := TERMINATOR;
    Error := SendAT6400Block(Address, Command);
    CharPtr := 0;
    Command[CharPtr] := TERMINATOR;
end;

(* ----- HIGH LEVEL GENERAL PURPOSE CONTROL PROCEDURES ----- *)
(* initialise the drives by reading a sequence of commands from an initialisation
file *)
procedure TStep.Initialise(NewAddress:word; readfilename:string);
var initfile : textfile;
    temp : string;
begin
    Address := NewAddress;
    AssignFile(initfile, readfilename);
    Reset(initfile);
    while not SeekEOF(initfile) do
        begin
            // read a command and then process it
            ReadLn(initfile, temp);
            ProcessCommand(temp);
        end;
    CloseFile(initfile);
end;

(* general purpose procedure for initiating drive movement at a fixed velocity
and in a certain direction *)
procedure TStep.GoDrive(drive1: Boolean; drive2: Boolean; drive3: Boolean;
    vel1: integer; vel2: integer; vel3: integer; dir1: TDirection;
    dir2: TDirection; dir3: TDirection);
var temp : string;
    dirtemp1 : string;
    dirtemp2 : string;
    dirtemp3 : string;
begin
    // continuous movement
    temp := 'MC1111';
    ProcessCommand(temp);

    // set the velocities of the drives
    temp := 'V'+inttostr(vel1)+'+',inttostr(vel2)+'+',inttostr(vel3)+'+',10';

```

```

ProcessCommand(temp);

// set the directions of the drives
if dir1 = CW then
    dirtemp1 := '+'
else
    dirtemp1 := '-';
if dir2 = CW then
    dirtemp2 := '+'
else
    dirtemp2 := '-';
if dir3 = CW then
    dirtemp3 := '+'
else
    dirtemp3 := '-';

temp := 'D'+dirtemp1+' '+dirtemp2+' '+dirtemp3+'-';
ProcessCommand(temp);

// initiate movement on drive
if drive1 then
    dirtemp1 := '1'
else
    dirtemp1 := '0';
if drive2 then
    dirtemp2 := '1'
else
    dirtemp2 := '0';
if drive3 then
    dirtemp3 := '1'
else
    dirtemp3 := '0';
temp := 'GO'+dirtemp1+dirtemp2+dirtemp3+'0';
ProcessCommand(temp);
end;

(* issue a command to immediately stop the drive movement at a controlled
deceleration ramp*)
procedure TStep.StopDrive;
var temp : string;
begin
    temp := 'S';
    ProcessCommand(temp);
end;

(* set the current drive position as the absolute zero *)
procedure TStep.SetZeroPos;
var temp : string;
begin
    temp := 'PSET0,0,0,0';
    ProcessCommand(temp);
end;

(* disable the drives *)
procedure TStep.DisableDrives;
var temp : string;
begin
    temp := 'DRIVE0000';
    ProcessCommand(temp);
    temp := 'GO0000';
    ProcessCommand(temp);
end;

(* reset the drives *)
procedure TStep.ResetDrives;
var temp : string;
begin
    temp := 'RESET';
    ProcessCommand(temp);
end;

(* move the drive continuously at a fixed velocity in a specific direction *)
procedure TStep.JogMove(drivenum: integer; dir: TDirection);
var drive1, drive2, drive3: Boolean;
    dir1, dir2, dir3 : TDirection;
    vel1, vel2, vel3 : integer;
begin
    drive1 := FALSE;
    drive2 := FALSE;
    drive3 := FALSE;
    vel1 := 1;
    vel2 := 1;
    vel3 := 1;
    dir1 := CW;
    dir2 := CW;
    dir3 := CW;

    // set the drive direction and enable drive
    case drivenum of
        1: begin drive1 := TRUE; dir1 := dir; end;
        2: begin drive2 := TRUE; dir2 := dir; end;
        3: begin drive3 := TRUE; dir3 := dir; end;
    end;
    GoDrive(drive1,drive2,drive3,vel1,vel2,vel3,dir1,dir2,dir3);
end;

procedure TStep.MoveFixedDist(vel1:real;vel2:real;vel3:real;
    dist1:integer;dist2:integer;dist3:integer);
var temp: string;
begin
    // set motion to move a fixed distance and to work according
    // to absolute co-ordinates
    temp := 'MC0000';
    ProcessCommand(temp);
    temp := 'MA0000';
    ProcessCommand(temp);

    // set the velocity
    temp := 'V'+floattostrf(vel1,ffGeneral,5,2)+' '+floattostrf(vel2,ffGeneral,5,2)
        +' '+floattostrf(vel3,ffGeneral,5,2)+' 0';
    ProcessCommand(temp);

    // set the distance
    temp := 'D'+inttostr(dist1)+' '+inttostr(dist2)+' '+inttostr(dist3)+' 0';
    ProcessCommand(temp);

    // initiate motion
    temp := 'GO1110';
    ProcessCommand(temp);
end;

(* get drive status from AT6400 *)
procedure TStep.GetMotorData(want: TRequestData; var axis1: double;
    var axis2: double; var axis3: double);
begin
    // instruct the AT6400 to update the fast status area
    RequestStatus;
    if want = Velocity then
        begin
            // get the drive velocity
            SetPointer(AXIS1_VELOCITY);
            ReadStatus(word_high, word_low, fast_status);
            axis1 := fast_status/400;
            ReadStatus(word_high, word_low, fast_status);
            axis2 := fast_status/400;
            ReadStatus(word_high, word_low, fast_status);
            axis3 := fast_status/400;
        end
    else if want = MotorPos then
        begin
            // get the drive position
            SetPointer(AXIS1_MOTOR);
            ReadStatus(word_high, word_low, fast_status);
            axis1 := fast_status;
            ReadStatus(word_high, word_low, fast_status);
            axis2 := fast_status;
            ReadStatus(word_high, word_low, fast_status);
            axis3 := fast_status;
        end;
    end;
end;

(* download the operating system *)
procedure TStep.DownloadOS;
begin
    WinExec('AT6400.EXE',SW_SHOWMINIMIZED);
end;

(* reset the drive limits *)
procedure TStep.ResetLimits;
var temp: string;
begin
    temp := '@LH0';

```

```

    ProcessCommand(temp);
    temp := '@LH3';
    ProcessCommand(temp);
end;

(* set the acceleration of the drives *)
procedure TStep.SetAcceleration(axis1, axis2, axis3: integer);
var temp: string;
begin
    temp := 'A'+inttostr(axis1)+' '+inttostr(axis2)+' '+inttostr(axis3)+' '+10';
    ProcessCommand(temp);
    temp := 'AD'+inttostr(axis1)+' '+inttostr(axis2)+' '+inttostr(axis3)+' '+10';
    ProcessCommand(temp);
end;

end.

```

PROGRAM LISTING OF DYNUNIT.PAS

(* This unit provides a means of verifying the dynamic results captured during the real-time testing. It operates purely in terms of volume fraction. Using a defined starting position for the bubble and knowledge of the bubble velocity and acceleration, it is able to predict when the bubble should be passing through a measurement section (since the position of the measurement electrodes is specified according to the bubble starting position). Since it can predict fairly accurately the position of a bubble at a certain time, it is therefore also able to predict the desired volume fractions of the different phases at that time. These desired volume fraction profiles are then compared to those captured from the real-time testing and an error calculation is performed. To compensate for slippage between the belt and pulley of the bubble drives, the real-time volume fraction profile is shifted to coincide with the desired volume fraction profile. *)

```
unit dynunit;
```

```
interface
```

```
uses
```

```
Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
StdCtrls, ExtCtrls, Spin, Buttons, TeEngine, Series, TeeProcs, Chart, Registry,
Math, TeeFunc;
```

```
type
```

```
TDynVerify = class(TForm)
    Panel1: TPanel;
    Panel3: TPanel;
    Label1: TLabel;
    AirPhaseConfig: TGroupBox;
    GravPhaseConfig: TGroupBox;
    PlateConfig: TGroupBox;
    Label2: TLabel;
    Label3: TLabel;
    Label4: TLabel;
    Label5: TLabel;
    LowerSystemBottom: TEdit;
    LowerSystemTop: TEdit;
    UpperSystemBottom: TEdit;
    UpperSystemTop: TEdit;
    Label6: TLabel;
    Label7: TLabel;
    Label8: TLabel;
    AirDiameter: TEdit;
    AirLength: TEdit;
    GravelDiameter: TEdit;
    GravelLength: TEdit;
    Label9: TLabel;
    Label10: TLabel;
    Label11: TLabel;
    ShapePipeAir: TShape;
    AirShape1: TShape;
    AirShape3: TShape;
    AirShape2: TShape;
    RealExit: TSpeedButton;
    AirShape4: TShape;
    AirShape5: TShape;
    GravShape4: TShape;
    GravShape3: TShape;

```

```

    GravShape5: TShape;
    GravShape2: TShape;
    GravShape1: TShape;
    ShapePipeGrav: TShape;
    Label12: TLabel;
    Label13: TLabel;
    TopAirPlot: TChart;
    Series3: TLineSeries;
    Series4: TLineSeries;
    BotAirPlot: TChart;
    LineSeries1: TLineSeries;
    LineSeries2: TLineSeries;
    TopGravelPlot: TChart;
    LineSeries3: TLineSeries;
    LineSeries4: TLineSeries;
    BotGravelPlot: TChart;
    LineSeries5: TLineSeries;
    LineSeries6: TLineSeries;
    CalcDesired: TSpeedButton;
    RealSaveResults: TSpeedButton;
    SaveResultsDialog: TSaveDialog;
    VelSource: TRadioGroup;
    CalcVolError: TSpeedButton;
    ErrorResults: TPanel;
    Label14: TLabel;
    Label15: TLabel;
    Label16: TLabel;
    Label17: TLabel;
    Label18: TLabel;
    Label19: TLabel;
    TopAirVolError: TLabel;
    BotAirVolError: TLabel;
    TopGravVolError: TLabel;
    BotGravVolError: TLabel;
    CalcActualVel: TSpeedButton;
    TopAirDelay: TLabel;
    BotAirDelay: TLabel;
    Label20: TLabel;
    TopGravDelay: TLabel;
    BotGravDelay: TLabel;
    Series1: TBarSeries;
    Series2: TBarSeries;
    Series5: TBarSeries;
    Series6: TBarSeries;
    Label21: TLabel;
    Label22: TLabel;
    AirBubbleStartPosition: TEdit;
    GravelBubbleStartPosition: TEdit;
    FileCustomName: TCheckBox;
    Label23: TLabel;
    TopAirPeak: TLabel;
    BotAirPeak: TLabel;
    TopGravPeak: TLabel;
    BotGravPeak: TLabel;
    procedure AirShape1MouseDown(Sender: TObject; Button: TMouseButton;
    Shift: TShiftState; X, Y: Integer);
    procedure AirShape2MouseDown(Sender: TObject; Button: TMouseButton;
    Shift: TShiftState; X, Y: Integer);
    procedure RealExitClick(Sender: TObject);
    procedure AirShape3MouseDown(Sender: TObject; Button: TMouseButton;
    Shift: TShiftState; X, Y: Integer);
    procedure AirShape4MouseDown(Sender: TObject; Button: TMouseButton;
    Shift: TShiftState; X, Y: Integer);
    procedure AirShape5MouseDown(Sender: TObject; Button: TMouseButton;
    Shift: TShiftState; X, Y: Integer);
    procedure GravShape1MouseDown(Sender: TObject; Button: TMouseButton;
    Shift: TShiftState; X, Y: Integer);
    procedure GravShape2MouseDown(Sender: TObject; Button: TMouseButton;
    Shift: TShiftState; X, Y: Integer);
    procedure GravShape3MouseDown(Sender: TObject; Button: TMouseButton;
    Shift: TShiftState; X, Y: Integer);
    procedure GravShape4MouseDown(Sender: TObject; Button: TMouseButton;
    Shift: TShiftState; X, Y: Integer);
    procedure GravShape5MouseDown(Sender: TObject; Button: TMouseButton;
    Shift: TShiftState; X, Y: Integer);
    procedure FormShow(Sender: TObject);
    procedure RealSaveResultsClick(Sender: TObject);
    procedure CalcDesiredClick(Sender: TObject);
    procedure CalcVolErrorClick(Sender: TObject);
    procedure CalcActualVelClick(Sender: TObject);
private

```

```

desiredvolume : array[1..6000,1..4] of real;
    // desired volume fractions for both phases at
    // the top and bottom system
airdiam : real;    // diameter of air bubble
gravidiam : real;  // diameter of gravel bubble
topairerror : real; // error in air prediction for top system
topgravelerror : real; // error in gravel prediction for top system
botairerror : real; // error in air prediction for bottom system
botgravelerror : real; // error in gravel prediction for bottom system
airtopstart : integer; // parameters specifying width of desired air
airtopstop : integer; // and gravel pulse
airbotstart : integer;
airbotstop : integer;
graveltopstart : integer;
graveltopstop : integer;
gravelbotstart : integer;
gravelbotstop : integer;
airleng : real;    // length of air bubble
gravleng : real;   // length of gravel bubble
uppersystop : real; // position of electrodes for top system
uppersysbot : real;
lowersystop : real; // position of electrodes for bottom system
lowersysbot : real;
airtopmax : integer; // parameters used for shifting real-time
airtopdesiremax : integer; // data to coincide with desired data to
graveltopmax : integer; // compensate for slippage on pulley
graveltopdesiremax : integer;
airbotmax : integer;
airbotdesiremax : integer;
gravelbotmax : integer;
gravelbotdesiremax : integer;
airtopoffset : integer;
airbotoffset : integer;
graveltopoffset : integer;
gravelbotoffset : integer;
// calculate the position of the two bubbles based on the time
procedure CalculateDistance(time:real; var airtopposition:
    real; var gravtopposition : real);
procedure ResetColours;
procedure UpdateDisplay;
// calculate the desired volume fraction for the two phases based on
// the positions
procedure CalculateVolume(airtopposition:real;gravtopposition:real;
    var upperairvol:real; var uppergravvol:real;var lowerairvol:real;
    var lowergravvol:real);
public
{ Public declarations }
airposition : integer; // position of air bubble
gravposition : integer; // position of gravel bubble
end;

var
DynVerify: TDynVerify;

implementation

uses realunit, stepconfig;

{$R *.DFM}

(* ----- DESIRED VOLUME FRACTION ESTIMATION PROCEDURES ----- *)
(* the following procedure calculates the position of the two bubbles at a
certain time using the velocity and accelerations stipulated in the stepper
configuration form. The position of the bubble is defined to be the top of
the bubble. Near the base of the pipeline there is a horizontal band defined
to be position zero. At the start of a test, the top of the bubble must coincide
with this band. As the bubble moves up the pipe, so the position (stipulated
in metres from the start band) increases. The positions of the measurement
electrodes are also provided with respect to this start band. *)
procedure TDynVerify.CalculateDistance(time: real; var airtopposition,
    gravtopposition: real);
var airfinaccel, gravfinaccel, airtimeflat, gravtimeflat : real;
    airtottime, gravtottime, tempairrps, tempgravrps, mintottime : real;
begin
    if VelSource.ItemIndex = 0 then
        begin
            tempairrps := StepperConfig.airrps;
            tempgravrps := StepperConfig.gravrps;
        end
    else
        begin
            if RealMain.airvelocity <> -1 then
                tempairrps :=
                    RealMain.airvelocity/StepperConfig.aircircumference
            else
                tempairrps := 0;
            if RealMain.gravelvelocity <> -1 then
                tempgravrps :=
                    RealMain.gravelvelocity/StepperConfig.gravelcircumference
            else
                tempgravrps := 0;
        end;

        // the position of the bubble is obtained by segmenting the path into three
        // separate sections, namely acceleration, constant velocity and deceleration
        // with StepperConfig do
        begin
            // calculate the position of the air bubble
            if tempairrps > 0 then
                begin
                    // time taken for bubble to accelerate to desired velocity
                    airfinaccel := tempairrps/airacceleration;
                    // time during which bubble moves at constant velocity
                    airtimeflat := (airrevs/tempairrps) - airfinaccel;
                    // total time
                    airtottime := airfinaccel*2 + airtimeflat;
                    mintottime := 2*sqrt(airrevs/airacceleration);

                    if airfinaccel > 0.5*mintottime then
                        begin
                            // if acceleration not high enough to reach constant velocity
                            if time < 0.5*mintottime then
                                airtopposition := 0.5*airacceleration*power(time,2)
                            else if time < mintottime then
                                airtopposition := 0.25*airacceleration*power(mintottime,2)
                                    -0.5*airacceleration*power(mintottime-time,2)
                            else
                                airtopposition := 0.25*airacceleration*power(mintottime,2);
                        end
                    else
                        begin
                            // during acceleration
                            if time < airfinaccel then
                                airtopposition := 0.5*airacceleration*power(time,2)
                            // during constant velocity
                            else if time < (airfinaccel+airtimeflat) then
                                airtopposition := 0.5*airfinaccel*tempairrps
                                    + (time-airfinaccel)*tempairrps
                            // during deceleration
                            else if time < airtottime then
                                airtopposition := airfinaccel*tempairrps + airtimeflat*tempairrps -
                                    0.5*airacceleration*power((airtottime-time),2)
                            else
                                airtopposition := airfinaccel*tempairrps + airtimeflat*tempairrps;
                        end;
                end;
            // convert revolutions into metres by multiplying by the circumference
            // of the pulley
            airtopposition := airtopposition*aircircumference;
        end
        else
            airtopposition := 0;

            // calculate the position of the gravel bubble
            if tempgravrps > 0 then
                begin
                    // same calculations are performed as for the air bubble except the
                    // gravel bubble parameters are used
                    gravfinaccel := tempgravrps/gravacceleration;
                    gravtimeflat := (gravrevs/tempgravrps) - gravfinaccel;
                    gravtottime := gravfinaccel*2 + gravtimeflat;
                    mintottime := 2*sqrt(gravrevs/gravacceleration);

                    if gravfinaccel > 0.5*mintottime then
                        begin
                            if time < 0.5*mintottime then
                                gravtopposition := 0.5*gravacceleration*power(time,2)
                            else if time < mintottime then
                                gravtopposition := 0.25*gravacceleration*power(mintottime,2)-
                                    0.5*gravacceleration*power(mintottime-time,2)
                            else
                                gravtopposition := 0.25*gravacceleration*power(mintottime,2);
                        end
                    else
                        begin
                            // during acceleration
                            if time < gravfinaccel then
                                gravtopposition := 0.5*gravacceleration*power(time,2)
                            // during constant velocity
                            else if time < (gravfinaccel+gravtimeflat) then
                                gravtopposition := 0.5*gravfinaccel*tempgravrps
                                    + (time-gravfinaccel)*tempgravrps
                            // during deceleration
                            else if time < gravtottime then
                                gravtopposition := gravfinaccel*tempgravrps + gravtimeflat*tempgravrps -
                                    0.5*gravacceleration*power((gravtottime-time),2)
                            else
                                gravtopposition := gravfinaccel*tempgravrps + gravtimeflat*tempgravrps;
                        end;
                end;
            // convert revolutions into metres by multiplying by the circumference
            // of the pulley
            gravtopposition := gravtopposition*gravcircumference;
        end
        else
            gravtopposition := 0;
    end;
end;

```

```

else
begin
if time < gravinaccel then
gravtopposition := 0.5*gravacceleration*power(time,2)
else if time < (gravinaccel+gravtimeflat) then
gravtopposition := 0.5*gravinaccel*tempgravrps
+ (time-gravinaccel)*tempgravrps
else if time < gravtottime then
gravtopposition := gravinaccel*tempgravrps
+gravtimeflat*tempgravrps-
0.5*gravacceleration*power((gravtottime-time),2)
else
gravtopposition := gravinaccel*tempgravrps
+ gravtimeflat*tempgravrps;
end;
gravtopposition := gravtopposition*gravcircumference;
end
else
gravtopposition := 0;
end;
end;

```

(* calculate the desired volume fractions and add the traces to the charts-the time is calculated using the estimated frame rate from the real-time unit since the start of the sampling and the start of the stepper motors occurs at the same time. The volume fractions are calculated as the volume of the bubble currently within the measuring region divided by the total volume of the measuring region. This is not the same as the component fraction if the bubble is shorter than the electrode length. *)

```

procedure TDynVerify.CalcDesiredClick(Sender: TObject);
var testposition : integer;
    airtopposition, gravtopposition, time : real;
    upperairvol, uppergravvol, lowerairvol, lowergravvol : real;
    upperplatevolume, lowerplatevolume : real;
    airstartpos, gravelstartpos : real;
begin
// initialise the charts
TopAirPlot.Series[0].Clear;
BotAirPlot.Series[0].Clear;
TopGravelPlot.Series[0].Clear;
BotGravelPlot.Series[0].Clear;
// obtain the air and gravel bubble parameters
airdiam := strtoint(AirDiameter.Text);
airleng := strtoint(AirLength.Text);
gravidiam := strtoint(GravelDiameter.Text);
gravleng := strtoint(GravelLength.Text);
uppersystop := strtoint(UpperSystemTop.Text);
uppersysbot := strtoint(UpperSystemBottom.Text);
lowersystop := strtoint(LowerSystemTop.Text);
lowersysbot := strtoint(LowerSystemBottom.Text);
airstartpos := strtoint(AirBubbleStartPosition.Text);
gravelstartpos := strtoint(GravelBubbleStartPosition.Text);
// calculate the volumes of the two measuring regions given that
// the pipe diameter is 350mm and the electrode length is given
// by the difference between uppersystop and uppersysbot
upperplatevolume := pi*power(0.175,2)*(uppersystop-uppersysbot);
lowerplatevolume := pi*power(0.175,2)*(lowersystop-lowersysbot);

```

```

for testposition := 1 to RealMain.DesiredFrames.Value do
begin

```

```

// calculate the estimated time of a particular frame as
// the frame number divided by the frame rate (or multiplied
// by the frame interval)
time := testposition * RealMain.frameinterval;
// calculate the position of the bubbles at this time
CalculateDistance(time,airtopposition,gravtopposition);
if StepperConfig.FlowDirSelect.ItemIndex = 0 then
begin
    airtopposition := airstartpos - airtopposition;
    gravtopposition := gravelstartpos - gravtopposition;
end
else
begin
    airtopposition := airtopposition + airstartpos;
    gravtopposition := gravtopposition + gravelstartpos;
end;
// based on the positions of the bubbles, calculate the corresponding
// volume fractions
CalculateVolume(airtopposition,gravtopposition,upperairvol,
    uppergravvol,lowerairvol,lowergravvol);
if StepperConfig.AddPhaseSelect.ItemIndex = 0 then

```

```

begin
    desiredvolume[testposition,1] := 100*(upperairvol+uppergravvol)
        /upperplatevolume;
    desiredvolume[testposition,2] := 0;
    desiredvolume[testposition,3] := 100*(lowerairvol+lowergravvol)
        /lowerplatevolume;
    desiredvolume[testposition,4] := 0;
end
else
begin
    desiredvolume[testposition,1] := 100*upperairvol/upperplatevolume;
    desiredvolume[testposition,2] := 100*uppergravvol/upperplatevolume;
    desiredvolume[testposition,3] := 100*lowerairvol/lowerplatevolume;
    desiredvolume[testposition,4] := 100*lowergravvol/lowerplatevolume;
end;
TopAirPlot.Series[0].AddY(desiredvolume[testposition,1]);
TopGravelPlot.Series[0].AddY(desiredvolume[testposition,2]);
BotAirPlot.Series[0].AddY(desiredvolume[testposition,3]);
BotGravelPlot.Series[0].AddY(desiredvolume[testposition,4]);
end
end;

```

(* calculate the volume of the bubble within the measuring regions based on the position of the bubble. Note that this algorithm makes various assumptions regarding the shape of the bubble, namely that the bubble is composed of a cylinder between two hemispheres and the length of the cylindrical section is greater than or equal to the length of the measuring section electrodes. The equations below are simply standard volume calculations obtained from tables and will not be discussed in any major detail. *)

```

procedure TDynVerify.CalculateVolume(airtopposition, gravtopposition: real;
    var upperairvol, uppergravvol, lowerairvol, lowergravvol: real);
var tempheight,airhemivol,gravhemivol : real;
begin
    airhemivol := (2/3)*pi*power(0.5*airdiam,3);
    gravhemivol := (2/3)*pi*power(0.5*gravidiam,3);

// air bubble volume for bottom measuring ring
// as round at top of bubble enters measuring section
if airleng > (lowersystop - lowersysbot) then
begin
    if(airtopposition>lowersysbot)
    and(airtopposition<=(lowersysbot+0.5*airdiam))then
    begin
        tempheight := lowersysbot - (airtopposition - 0.5*airdiam);
        lowerairvol := airhemivol-(pi*tempheight*power(0.5*airdiam,2)-
            (1/3)*pi*power(tempheight,3));
    end
    // as cylinder section of bubble enters measuring ring
    else if(airtopposition>(lowersysbot+0.5*airdiam))
    and(airtopposition<=lowersystop)then
    begin
        tempheight := (airtopposition-0.5*airdiam)-lowersysbot;
        lowerairvol := airhemivol+pi*tempheight*power(0.5*airdiam,2);
    end
    // as round at top of bubble exits measuring section
    else if(airtopposition>lowersystop)
    and(airtopposition<=(lowersystop+0.5*airdiam))then
    begin
        tempheight := lowersystop-(airtopposition-0.5*airdiam);
        lowerairvol := (pi*tempheight*power(0.5*airdiam,2)
            -(1/3)*pi*power(tempheight,3))+pi*(lowersystop-lowersysbot-tempheight)
            *power(0.5*airdiam,2);
    end
    // as only the cylindrical portion of the bubble remains in the measuring ring
    else if(airtopposition>(lowersystop+0.5*airdiam))
    and((airtopposition-airleng+0.5*airdiam)<=lowersysbot)then
    begin
        lowerairvol := pi*(lowersystop-lowersysbot)*power(0.5*airdiam,2);
    end
    // as round at bottom of bubble enters measuring section
    else if((airtopposition-airleng+0.5*airdiam)>lowersysbot)
    and((airtopposition-airleng+0.5*airdiam)<=(lowersysbot+0.5*airdiam))then
    begin
        tempheight := (airtopposition-airleng+0.5*airdiam)-lowersysbot;
        lowerairvol := (pi*tempheight*power(0.5*airdiam,2)-(1/3)
            *pi*power(tempheight,3))+pi*(lowersystop-lowersysbot-tempheight)
            *power(0.5*airdiam,2);
    end
    // as round at bottom of the bubble is fully inside the measuring section
    else if((airtopposition-airleng+0.5*airdiam)>(lowersysbot+0.5*airdiam))
    and((airtopposition-airleng+0.5*airdiam)<=lowersystop)then

```



```

else if (gravtopposition > lowersystop)
and((gravtopposition-gravleng)<=lowersystop)then
  lowergravvol := pi*power(0.5*gravdiam,2)*(lowersystop-
    (gravtopposition-gravleng))
else
  lowergravvol := 0;
end;

// gravel bubble for top measuring ring
if gravleng > (uppersystop-uppersysbot) then
begin
  if(gravtopposition>uppersysbot)and
  (gravtopposition<=(uppersysbot+0.5*gravdiam))then
  begin
    tempheight := uppersysbot - (gravtopposition - 0.5*gravdiam);
    uppergravvol := gravhemivol-(pi*tempheight*power(0.5*gravdiam,2)-
      (1/3)*pi*power(tempheight,3));
  end
  else if(gravtopposition>(uppersysbot+0.5*gravdiam))
  and(gravtopposition<=uppersystop)then
  begin
    tempheight := (gravtopposition-0.5*gravdiam)-uppersysbot;
    uppergravvol := gravhemivol+pi*tempheight*power(0.5*gravdiam,2);
  end
  else if(gravtopposition>uppersystop)
  and(gravtopposition<=(uppersystop+0.5*gravdiam))then
  begin
    tempheight := uppersystop-(gravtopposition-0.5*gravdiam);
    uppergravvol := (pi*tempheight*power(0.5*gravdiam,2)-(1/3)
      *pi*power(tempheight,3))+pi*(uppersystop-uppersysbot-tempheight)
      *power(0.5*gravdiam,2);
  end
  else if(gravtopposition>(uppersystop+0.5*gravdiam))
  and((gravtopposition-gravleng+0.5*gravdiam)<=uppersysbot)then
  begin
    uppergravvol := pi*(uppersystop-uppersysbot)*power(0.5*gravdiam,2);
  end
  else if((gravtopposition-gravleng+0.5*gravdiam)>uppersysbot)
  and((gravtopposition-gravleng+0.5*gravdiam)<=(uppersysbot+0.5*gravdiam))then
  begin
    tempheight := (gravtopposition-gravleng+0.5*gravdiam)-uppersysbot;
    uppergravvol := (pi*tempheight*power(0.5*gravdiam,2)-(1/3)
      *pi*power(tempheight,3))+pi*(uppersystop-uppersysbot-tempheight)
      *power(0.5*gravdiam,2);
  end
  else if((gravtopposition-gravleng+0.5*gravdiam)>(uppersysbot+0.5*gravdiam))
  and((gravtopposition-gravleng+0.5*gravdiam)<=uppersystop)then
  begin
    tempheight := uppersystop-(gravtopposition-gravleng+0.5*gravdiam);
    uppergravvol := gravhemivol + pi*tempheight*power(0.5*gravdiam,2);
  end
  else if ((gravtopposition-gravleng+0.5*gravdiam)>uppersystop)
  and((gravtopposition-gravleng)<=uppersystop)then
  begin
    tempheight := (gravtopposition-gravleng+0.5*gravdiam)-uppersystop;
    uppergravvol := gravhemivol - (pi*tempheight*power(0.5*gravdiam,2)-
      (1/3)*pi*power(tempheight,3));
  end
  else
    uppergravvol := 0;
  end
end
else
begin
  if (gravtopposition > uppersysbot)
  and((gravtopposition-gravleng) <= uppersysbot)then
    uppergravvol := pi*power(0.5*gravdiam,2)*(gravtopposition-uppersysbot)
  else if ((gravtopposition-gravleng) > uppersysbot)
  and(gravtopposition <= uppersystop)then
    uppergravvol := pi*power(0.5*gravdiam,2)*gravleng
  else if (gravtopposition > uppersystop)
  and((gravtopposition-gravleng)<=uppersystop)then
    uppergravvol := pi*power(0.5*gravdiam,2)*(uppersystop-
      (gravtopposition-gravleng))
  else
    uppergravvol := 0;
  end;
end;
end;

```

(* this procedure calculates the mean absolute error between the desired volume and the real-time volume fraction data. To compensate for slippage between the belt and pulley of the stepper motors, the real-time volume fraction data is

shifted to coincide with the desired volume fraction data. As the bubble passes through the measuring section, a bell-shaped volume fraction peak is created. To improve the accuracy of the assessment, the mean error is only calculated over the duration of this peak. By detecting the peak of both the desired volume fraction data and the real-time predicted data, the data can be shifted so that the peaks coincide and any error as a result of slippage will be corrected. *)
 procedure TDynVerify.CalcVolErrorClick(Sender: TObject);
 var testcase, startmax, stopmax: integer;
 maxvalue : real;

```

begin
  // initialise variables
  topairerror := 0;
  topgravelerror := 0;
  botairerror := 0;
  botgravelerror := 0;
  airtopdesiremax := -1;
  graveltopdesiremax := -1;
  airbotdesiremax := -1;
  gravelbotdesiremax := -1;
  airtopstart := 0;
  airtopstop := 0;
  airbotstart := 0;
  airbotstop := 0;
  graveltopstart := 0;
  graveltopstop := 0;
  gravelbotstart := 0;
  gravelbotstop := 0;
  ErrorResults.Visible := true;

  startmax := 0;
  stopmax := 0;
  maxvalue := 0;
  // air bubble at top measuring ring
  // firstly find the peak of the real-time predicted volume fraction data
  for testcase := 1 to RealMain.DesiredFrames.Value do
  begin
    if totaloutputs = 2 then
    begin
      if outputstore[testcase].prediction[1] > maxvalue then
      begin
        maxvalue := outputstore[testcase].prediction[1];
        startmax := testcase;
        stopmax := 0;
      end
      else if outputstore[testcase].prediction[1] = maxvalue then
        stopmax := testcase;
      end
    end
    else
    begin
      if volumestore[testcase,1] > maxvalue then
      begin
        maxvalue := volumestore[testcase,1];
        startmax := testcase;
        stopmax := 0;
      end
      else if volumestore[testcase,1] = maxvalue then
        stopmax := testcase;
      end
    end
  end;
  TopAirPeak.Caption := floattostrf(maxvalue,ffFixed,4,3);
  if stopmax <> 0 then
    airtopmax := floor(startmax + 0.5*(stopmax-startmax))
  else
    airtopmax := startmax;

  // then calculate the desired width of the volume fraction peak
  // and the position of the desired volume fraction peak
  maxvalue := 0;
  startmax := 0;
  stopmax := 0;
  for testcase := 1 to RealMain.DesiredFrames.Value do
  begin
    if desiredvolume[testcase,1] > maxvalue then
    begin
      maxvalue := desiredvolume[testcase,1];
      startmax := testcase;
      stopmax := 0;
    end
    else if desiredvolume[testcase,1] = maxvalue then
      stopmax := testcase;
    end
  end
end;

```

```

if (desiredvolume[testcase,1]> 0)and(airtopstart = 0)then
    airtopstart := testcase
else if (desiredvolume[testcase,1]> 0)and(airtopstart <> 0)then
    airtopstop := testcase;
end;
if maxvalue = 0 then
    airtopdesiremax := -1
else if stopmax <> 0 then
    airtopdesiremax := floor(startmax + 0.5*(stopmax-startmax))
else
    airtopdesiremax := startmax;

// gravel volume for top measuring ring
startmax := 0;
stopmax := 0;
maxvalue := 0;
for testcase := 1 to RealMain.DesiredFrames.Value do
begin
    if totaloutputs = 2 then
        begin
            if outputstore[testcase].prediction[2] > maxvalue then
                begin
                    maxvalue := outputstore[testcase].prediction[2];
                    startmax := testcase;
                    stopmax := 0;
                end
            else if outputstore[testcase].prediction[2] = maxvalue then
                stopmax := testcase;
            end
        end
    else
        begin
            if volumestore[testcase,2] > maxvalue then
                begin
                    maxvalue := volumestore[testcase,2];
                    startmax := testcase;
                    stopmax := 0;
                end
            else if volumestore[testcase,2] = maxvalue then
                stopmax := testcase;
            end
        end
    end;
    TopGravPeak.Caption := floattostrf(maxvalue,ffFixed,4,3);
    if stopmax <> 0 then
        graveltopmax := floor(startmax + 0.5*(stopmax-startmax))
    else
        graveltopmax := startmax;

    maxvalue := 0;
    startmax := 0;
    stopmax := 0;
    for testcase := 1 to RealMain.DesiredFrames.Value do
    begin
        if desiredvolume[testcase,2] > maxvalue then
            begin
                maxvalue := desiredvolume[testcase,2];
                startmax := testcase;
                stopmax := 0;
            end
        else if desiredvolume[testcase,2] = maxvalue then
            stopmax := testcase;

        if (desiredvolume[testcase,2] > 0)and(graveltopstart = 0)then
            graveltopstart := testcase
        else if (desiredvolume[testcase,2] > 0)and(graveltopstart <> 0)then
            graveltopstop := testcase;
        end;
        if maxvalue = 0 then
            graveltopdesiremax := -1
        else if stopmax <> 0 then
            graveltopdesiremax := floor(startmax + 0.5*(stopmax-startmax))
        else
            graveltopdesiremax := startmax;

        topairerror := 0;
        airtopoffset := airtopmax-airtopdesiremax;
        graveltopoffset := graveltopmax-graveltopdesiremax;
        // shift the data and calculate the mean absolute error between the
        // desired and predicted volume fraction data over the width of the peak
        // where MAE = mean(abs(error))
        if airtopdesiremax <> -1 then
            begin

```

```

// calculate the time delay between the desired volume fraction
// peak and the real-time peak - if there was no slippage (for
// example, a synchronous belt was used), then this would give
// a good indication of the time response of the system electronics
TopAirDelay.Caption :=
    floattostrf(RealMain.frameinterval*airtopoffset,ffFixed,4,3);
for testcase := airtopstart to airtopstop do
begin
    if totaloutputs = 2 then
        topairerror := topairerror+abs(desiredvolume[testcase,1]-
            outputstore[testcase+airtopoffset].prediction[1])
    else
        topairerror := topairerror+abs(desiredvolume[testcase,1]-
            volumestore[testcase+airtopoffset,1]);
    end;
    topairerror := topairerror/(airtopstop-airtopstart+1);
end
// if the algorithm is unable to determine a peak then simply calculate
// the mean absolute error over all the width of the opposite phase
else
begin
    TopAirDelay.Caption := 'unknown';
    for testcase := graveltopstart to graveltopstop do
    begin
        if totaloutputs = 2 then
            topairerror := topairerror+abs(0-
                outputstore[testcase+graveltopoffset].prediction[1])
        else
            topairerror := topairerror+abs(0-
                volumestore[testcase+graveltopoffset,1]);
        end;
        topairerror := topairerror/(graveltopstop-graveltopstart+1);
    end;
    TopAirVolError.Caption := floattostrf(topairerror,ffFixed,4,3);

    topgravelerror := 0;
    if graveltopdesiremax <> -1 then
    begin
        TopGravDelay.Caption :=
            floattostrf(RealMain.frameinterval*graveltopoffset,ffFixed,4,3);
        for testcase := graveltopstart to graveltopstop do
        begin
            if totaloutputs = 2 then
                topgravelerror := topgravelerror+abs(desiredvolume[testcase,2]-
                    outputstore[testcase+graveltopoffset].prediction[2])
            else
                topgravelerror := topgravelerror+abs(desiredvolume[testcase,2]-
                    volumestore[testcase+graveltopoffset,2]);
            end;
            topgravelerror := topgravelerror/(graveltopstop-graveltopstart+1);
        end
    else
    begin
        TopGravDelay.Caption := 'unknown';
        for testcase := airtopstart to airtopstop do
        begin
            if totaloutputs = 2 then
                topgravelerror := topgravelerror+abs(0-
                    outputstore[testcase+airtopoffset].prediction[2])
            else
                topgravelerror := topgravelerror+abs(0-
                    volumestore[testcase+airtopoffset,2]);
            end;
            topgravelerror := topgravelerror/(airtopstop-airtopstart+1);
        end;
        TopGravVolError.Caption := floattostrf(topgravelerror,ffFixed,4,3);

        // air volume fraction for bottom measuring ring
        startmax := 0;
        stopmax := 0;
        maxvalue := 0;
        for testcase := 1 to RealMain.DesiredFrames.Value do
        begin
            if totaloutputs = 2 then
                begin
                    if outputstore[testcase].prediction[3] > maxvalue then
                        begin
                            maxvalue := outputstore[testcase].prediction[3];
                            startmax := testcase;
                            stopmax := 0;
                        end
                    end
                end
            end
        end
    end
end

```

```

        else if outputstore[testcase].prediction[3] = maxvalue then
            stopmax := testcase;
        end
    else
        begin
            if volumestore[testcase,3] > maxvalue then
                begin
                    maxvalue := volumestore[testcase,3];
                    startmax := testcase;
                    stopmax := 0;
                end
            else if volumestore[testcase,3] = maxvalue then
                stopmax := testcase;
            end
        end;
        BotAirPeak.Caption := floattostrf(maxvalue,ffFixed,4,3);
        if stopmax <> 0 then
            airbotmax := floor(startmax + 0.5*(stopmax-startmax))
        else
            airbotmax := startmax;

        maxvalue := 0;
        startmax := 0;
        stopmax := 0;
        for testcase := 1 to RealMain.DesiredFrames.Value do
            begin
                if desiredvolume[testcase,3] > maxvalue then
                    begin
                        maxvalue := desiredvolume[testcase,3];
                        startmax := testcase;
                        stopmax := 0;
                    end
                else if desiredvolume[testcase,3] = maxvalue then
                    stopmax := testcase;
                end

                if (desiredvolume[testcase,3] > 0) and (airbotstart = 0) then
                    airbotstart := testcase
                else if (desiredvolume[testcase,3] > 0) and (airbotstart <> 0) then
                    airbotstop := testcase;
                end;
            end
            if maxvalue = 0 then
                airbotdesiremax := -1
            else if stopmax <> 0 then
                airbotdesiremax := floor(startmax + 0.5*(stopmax-startmax))
            else
                airbotdesiremax := startmax;

            // gravel volume fraction for bottom measuring ring
            startmax := 0;
            stopmax := 0;
            maxvalue := 0;
            for testcase := 1 to RealMain.DesiredFrames.Value do
                begin
                    if totaloutputs = 2 then
                        begin
                            if outputstore[testcase].prediction[4] > maxvalue then
                                begin
                                    maxvalue := outputstore[testcase].prediction[4];
                                    startmax := testcase;
                                    stopmax := 0;
                                end
                            else if outputstore[testcase].prediction[4] = maxvalue then
                                stopmax := testcase;
                            end
                        end
                    else
                        begin
                            if volumestore[testcase,4] > maxvalue then
                                begin
                                    maxvalue := volumestore[testcase,4];
                                    startmax := testcase;
                                    stopmax := 0;
                                end
                            else if volumestore[testcase,4] = maxvalue then
                                stopmax := testcase;
                            end
                        end
                    end;
                end
                BotGravPeak.Caption := floattostrf(maxvalue,ffFixed,4,3);
                if stopmax <> 0 then
                    gravelbotmax := floor(startmax + 0.5*(stopmax-startmax))
                else

```

```

                    gravelbotmax := startmax;

                    maxvalue := 0;
                    startmax := 0;
                    stopmax := 0;
                    for testcase := 1 to RealMain.DesiredFrames.Value do
                        begin
                            if desiredvolume[testcase,4] > maxvalue then
                                begin
                                    maxvalue := desiredvolume[testcase,4];
                                    startmax := testcase;
                                    stopmax := 0;
                                end
                            else if desiredvolume[testcase,4] = maxvalue then
                                stopmax := testcase;
                            end

                            if (desiredvolume[testcase,4] > 0) and (gravelbotstart = 0) then
                                gravelbotstart := testcase
                            else if (desiredvolume[testcase,4] > 0) and (gravelbotstart <> 0) then
                                gravelbotstop := testcase;
                            end;
                        end
                        if maxvalue = 0 then
                            gravelbotdesiremax := -1
                        else if stopmax <> 0 then
                            gravelbotdesiremax := floor(startmax + 0.5*(stopmax-startmax))
                        else
                            gravelbotdesiremax := startmax;

                        botairerror := 0;
                        airbotoffset := airbotmax-airbotdesiremax;
                        gravelbotoffset := gravelbotmax-gravelbotdesiremax;
                        if airbotdesiremax <> -1 then
                            begin
                                BotAirDelay.Caption :=
                                    floattostrf(RealMain.frameinterval*airbotoffset,ffFixed,4,3);
                                for testcase := airbotstart to airbotstop do
                                    begin
                                        if totaloutputs = 2 then
                                            botairerror:=botairerror+abs(desiredvolume[testcase,3]-
                                                outputstore[testcase+airbotoffset].prediction[3])
                                        else
                                            botairerror:=botairerror+abs(desiredvolume[testcase,3]-
                                                volumestore[testcase+airbotoffset,3]);
                                        end;
                                    end
                                    botairerror := botairerror/(airbotstop-airbotstart+1);
                                end
                            end
                            else
                                begin
                                    BotAirDelay.Caption := 'unknown';
                                    for testcase := gravelbotstart to gravelbotstop do
                                        begin
                                            if totaloutputs = 2 then
                                                botairerror:=botairerror+abs(0-
                                                    outputstore[testcase+gravelbotoffset].prediction[3])
                                            else
                                                botairerror:=botairerror+abs(0-
                                                    volumestore[testcase+gravelbotoffset,3]);
                                            end;
                                        end
                                        botairerror := botairerror/(gravelbotstop-gravelbotstart+1);
                                    end
                                end
                                BotAirVolError.Caption := floattostrf(botairerror,ffFixed,4,3);

                            botgravelerror := 0;
                            if gravelbotdesiremax <> -1 then
                                begin
                                    BotGravDelay.Caption :=
                                        floattostrf(RealMain.frameinterval*gravelbotoffset,ffFixed,4,3);
                                    for testcase := gravelbotstart to gravelbotstop do
                                        begin
                                            if totaloutputs = 2 then
                                                botgravelerror:=botgravelerror+abs(desiredvolume[testcase,4]-
                                                    outputstore[testcase+gravelbotoffset].prediction[4])
                                            else
                                                botgravelerror:=botgravelerror+abs(desiredvolume[testcase,4]-
                                                    volumestore[testcase+gravelbotoffset,4]);
                                            end;
                                        end
                                        botgravelerror := botgravelerror/(gravelbotstop-gravelbotstart+1);
                                    end
                                end
                            end
                            else
                                begin
                                    BotGravDelay.Caption := 'unknown';

```

```

for testcase := airbotstart to airbotstop do
begin
  if totaloutputs = 2 then
    botgravelerror:=botgravelerror+abs(0-
      outputstore[testcase+airbotoffset].prediction[4])
  else
    botgravelerror:=botgravelerror+abs(0-
      volumestore[testcase+airbotoffset,4]);
  end;
  botgravelerror := botgravelerror/(airbotstop-airbotstart+1);
end;
BotGravVolError.Caption := floattostrf(botgravelerror,ffFixed,4,3);

// add markers to the chart to indicate what the algorithm determined
// to be the peaks of the different plots
if airtopdesiremax <> -1 then
begin
  TopAirPlot.Series[2].AddXY(airtopmax-1,10,",clBlue);
  TopAirPlot.Series[2].AddXY(airtopdesiremax-1,10,",clRed);
end;
if graveltopdesiremax <> -1 then
begin
  TopGravelPlot.Series[2].AddXY(graveltopmax-1,10,",clBlue);
  TopGravelPlot.Series[2].AddXY(graveltopdesiremax-1,10,",clRed);
end;
if airbotdesiremax <> -1 then
begin
  BotAirPlot.Series[2].AddXY(airbotmax-1,10,",clBlue);
  BotAirPlot.Series[2].AddXY(airbotdesiremax-1,10,",clRed);
end;
if gravelbotdesiremax <> -1 then
begin
  BotGravelPlot.Series[2].AddXY(gravelbotmax-1,10,",clBlue);
  BotGravelPlot.Series[2].AddXY(gravelbotdesiremax-1,10,",clRed);
end;
end;

(* ----- ACTUAL VELOCITY CALCULATION PROCEDURE ----- *)
(* since the stepper motors only provide a relatively low acceleration rate,
the bubble does not always reach constant speed before entering the measuring
rings. The cross-correlation algorithm calculates an average velocity between the
two measuring rings so if the bubble has not reached constant velocity before
entering the measuring section, the cross-correlation determined velocity
will not equal the desired velocity. This algorithm takes that into consideration
and adjusts the desired velocity using the same principles as the algorithm to
calculate the position of the bubble at a specific time. *)
procedure TDynVerify.CalcActualVelClick(Sender: TObject);
var simtime,airtoptime,gravtoptime,airbottime,gravbottime : real;
airtopposition,gravtopposition,airtransit,gravtransit : real;
begin
  airtoptime := 0;
  gravtoptime := 0;
  airbottime := 0;
  gravbottime := 0;
  simtime := 0;
  // to calculate the average velocity of the bubble between the
  // two measuring rings, find the time when the bubble passed the
  // centre of the first measuring ring, the time when the bubble
  // passed the centre of the second measuring ring and divide the
  // measuring ring separation by this transit time
  while(simtime < RealMain.DesiredFrames.Value*RealMain.frameinterval)do
  begin
    CalculateDistance(simtime,airtopposition,gravtopposition);
    // time when air bubble at centre of bottom measuring ring
    if ((airtopposition - 0.5*airleng)>=(lowersysbot
      +0.5*(lowersystop-lowersysbot)-0.005))and
      ((airtopposition - 0.5*airleng)<=(lowersysbot
      +0.5*(lowersystop-lowersysbot)+0.005))then
      airbottime := simtime;
    // time when air bubble at centre of top measuring ring
    if ((airtopposition - 0.5*airleng)>=(uppersysbot
      +0.5*(uppersystop-uppersysbot)-0.005))and
      ((airtopposition - 0.5*airleng)<=(uppersysbot
      +0.5*(uppersystop-uppersysbot)+0.005))then
      airttoptime := simtime;
    // time when gravel bubble at centre of bottom measuring ring
    if ((gravtopposition - 0.5*gravleng)>=(lowersysbot
      +0.5*(lowersystop-lowersysbot)-0.005))and
      ((gravtopposition - 0.5*gravleng)<=(lowersysbot
      +0.5*(lowersystop-lowersysbot)+0.005))then
      gravbottime := simtime;

```

```

    // time when gravel bubble at centre of top measuring ring
    if ((gravtopposition - 0.5*gravleng)>=(uppersysbot
      +0.5*(uppersystop-uppersysbot)-0.005))and
      ((gravtopposition - 0.5*gravleng)<=(uppersysbot
      +0.5*(uppersystop-uppersysbot)+0.005))then
      gravtoptime := simtime;
    simtime := simtime + 0.001;
  end;
  // calculate transit times of two bubbles and then the average velocity
  airtransit := abs(airtoptime-airbottime);
  gravtransit := abs(gravtoptime-gravbottime);
  with RealMain do
  begin
    if (airtransit = 0) then
      begin
        airactualvel := 0;
        AirVelActual.Caption := '0'
      end
    else
      begin
        airactualvel := (flowmeterfactor*systemseparation)/airtransit;
        if airactualvel < 0.001 then
          airactualvel := 0;
          AirVelActual.Caption := floattostrf(airactualvel,ffFixed,4,3);
        end;
        if (gravtransit = 0) then
          begin
            gravelactualvel := 0;
            GravelVelActual.Caption := '0';
          end
        else
          begin
            gravelactualvel := (flowmeterfactor*systemseparation)/gravtransit;
            if gravelactualvel < 0.001 then
              gravelactualvel := 0;
              GravelVelActual.Caption := floattostrf(gravelactualvel,ffFixed,4,3);
            end;
          end
        end;
      end;
    end;

    (* ----- GENERAL PROCEDURES ----- *)
    (* the following functions provide a simple graphical interface for
    specifying the position of the gravel and air bubble *)
    procedure TDynVerify.AirShape1MouseDown(Sender: TObject; Button: TMouseButton;
      Shift: TShiftState; X, Y: Integer);
    begin
      airposition := 1;
      UpdateDisplay;
    end;
    procedure TDynVerify.AirShape2MouseDown(Sender: TObject; Button: TMouseButton;
      Shift: TShiftState; X, Y: Integer);
    begin
      airposition := 2;
      UpdateDisplay;
    end;
    procedure TDynVerify.AirShape3MouseDown(Sender: TObject; Button: TMouseButton;
      Shift: TShiftState; X, Y: Integer);
    begin
      airposition := 3;
      UpdateDisplay;
    end;
    procedure TDynVerify.AirShape4MouseDown(Sender: TObject; Button: TMouseButton;
      Shift: TShiftState; X, Y: Integer);
    begin
      airposition := 4;
      UpdateDisplay;
    end;
    procedure TDynVerify.AirShape5MouseDown(Sender: TObject; Button: TMouseButton;
      Shift: TShiftState; X, Y: Integer);
    begin
      airposition := 5;
      UpdateDisplay;
    end;
    procedure TDynVerify.GravShape1MouseDown(Sender: TObject;
      Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
    begin
      gravelposition := 1;
      UpdateDisplay;
    end;
    procedure TDynVerify.GravShape2MouseDown(Sender: TObject;
      Button: TMouseButton; Shift: TShiftState; X, Y: Integer);

```

```

begin
    gravelposition := 2;
    UpdateDisplay;
end;
procedure TDynVerify.GravShape3MouseDown(Sender: TObject;
    Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
begin
    gravelposition := 3;
    UpdateDisplay;
end;
procedure TDynVerify.GravShape4MouseDown(Sender: TObject;
    Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
begin
    gravelposition := 4;
    UpdateDisplay;
end;
procedure TDynVerify.GravShape5MouseDown(Sender: TObject;
    Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
begin
    gravelposition := 5;
    UpdateDisplay;
end;
(* reset the graphical interface *)
procedure TDynVerify.ResetColours;
begin
    AirShape1.Brush.Color := clBlack;
    AirShape2.Brush.Color := clBlack;
    AirShape3.Brush.Color := clBlack;
    AirShape4.Brush.Color := clBlack;
    AirShape5.Brush.Color := clBlack;
    GravShape1.Brush.Color := clBlack;
    GravShape2.Brush.Color := clBlack;
    GravShape3.Brush.Color := clBlack;
    GravShape4.Brush.Color := clBlack;
    GravShape5.Brush.Color := clBlack;
end;
(* update the graphical interface indicating bubble position *)
procedure TDynVerify.UpdateDisplay;
begin
    ResetColours;
    case airposition of
        0: begin end;
        1: begin AirShape1.Brush.Color := clWhite; end;
        2: begin AirShape2.Brush.Color := clWhite; end;
        3: begin AirShape3.Brush.Color := clWhite; end;
        4: begin AirShape4.Brush.Color := clWhite; end;
        5: begin AirShape5.Brush.Color := clWhite; end;
    end;
    case gravelposition of
        0: begin end;
        1: begin GravShape1.Brush.Color := clGray; end;
        2: begin GravShape2.Brush.Color := clGray; end;
        3: begin GravShape3.Brush.Color := clGray; end;
        4: begin GravShape4.Brush.Color := clGray; end;
        5: begin GravShape5.Brush.Color := clGray; end;
    end;
end;
end;
(* when close form, save the current settings to the registry *)
procedure TDynVerify.RealExitClick(Sender: TObject);
var Reg : TRegistry;
begin
    Reg := TRegistry.Create;
    try
        Reg.OpenKey('Software\Neural', True);
        Reg.WriteString('AirDiameter', AirDiameter.Text);
        Reg.WriteString('AirLength', AirLength.Text);
        Reg.WriteString('AirBubbleStartPosition', AirBubbleStartPosition.Text);
        Reg.WriteString('GravelDiameter', GravelDiameter.Text);
        Reg.WriteString('GravelLength', GravelLength.Text);
    Reg.WriteString('GravelBubbleStartPosition', GravelBubbleStartPosition.Text);
        Reg.WriteInteger('AirPosition', airposition);
        Reg.WriteInteger('GravelPosition', gravelposition);
        Reg.WriteString('UpperSystTop', UpperSystemTop.Text);
        Reg.WriteString('UpperSystBot', UpperSystemBottom.Text);
        Reg.WriteString('LowerSystTop', LowerSystemTop.Text);
        Reg.WriteString('LowerSystBot', LowerSystemBottom.Text);
        Reg.WriteInteger('VelSource', VelSource.ItemIndex);
    finally
        Reg.Free;
    end;
end;

```

```

end;
Close;
end;
(* when form opens, initialise variables and load real-time volume fraction
data into charts *)
procedure TDynVerify.FormShow(Sender: TObject);
var testindex : integer;
begin
    if StepperConfig.airrps > 0 then
        AirPhaseConfig.Enabled := true
    else
        begin
            airposition := 0;
            AirPhaseConfig.Enabled := false;
        end;
    if StepperConfig.gravelrps > 0 then
        GravPhaseConfig.Enabled := true
    else
        begin
            gravelposition := 0;
            GravPhaseConfig.Enabled := false;
        end;
    ErrorResults.Visible := false;
    UpdateDisplay;
    TopAirPlot.Series[0].Clear;
    TopAirPlot.Series[1].Clear;
    TopAirPlot.Series[2].Clear;
    TopGravelPlot.Series[0].Clear;
    TopGravelPlot.Series[1].Clear;
    TopGravelPlot.Series[2].Clear;
    BotAirPlot.Series[0].Clear;
    BotAirPlot.Series[1].Clear;
    BotAirPlot.Series[2].Clear;
    BotGravelPlot.Series[0].Clear;
    BotGravelPlot.Series[1].Clear;
    BotGravelPlot.Series[2].Clear;
    for testindex := 1 to RealMain.DesiredFrames.Value do
        begin
            if totaloutputs = 2 then
                begin
                    // volume fraction data
                    TopAirPlot.Series[1].AddY(outputstore[testindex].prediction[1]);
                    TopGravelPlot.Series[1].AddY(outputstore[testindex].prediction[2]);
                    BotAirPlot.Series[1].AddY(outputstore[testindex].prediction[3]);
                    BotGravelPlot.Series[1].AddY(outputstore[testindex].prediction[4]);
                end
            else
                begin
                    // image data
                    TopAirPlot.Series[1].AddY(volumestore[testindex,1]);
                    TopGravelPlot.Series[1].AddY(volumestore[testindex,2]);
                    BotAirPlot.Series[1].AddY(volumestore[testindex,3]);
                    BotGravelPlot.Series[1].AddY(volumestore[testindex,4]);
                end;
            end;
            VelSource.SetFocus;
        end;
    end;
    (* save the results to a textfile so that they may be loaded in MATLAB for
analysis at a later stage *)
    procedure TDynVerify.RealSaveResultsClick(Sender: TObject);
    var tempfile: textfile;
        filename: string;
        tempstring : string;
        row,col,totalsecs: integer;
        hours,mins,secs,msecs : word;
        topairposition, topgravelposition : real;
    begin
        // order for output file is
        // sample time, interpolated time, air bubble position, gravel position
        // top air volume fraction, top gravel volume fraction, bottom air
        // volume fraction, bottom gravel fraction, desired top air fraction,
        // desired top gravel fraction, desired bottom air fraction, desired
        // bottom gravel fraction, air correlation, gravel correlation
        if FileCustomName.Checked then
            begin
                SaveResultsDialog.Execute;
                filename := SaveResultsDialog.FileName;
            end
        else
            begin

```

```

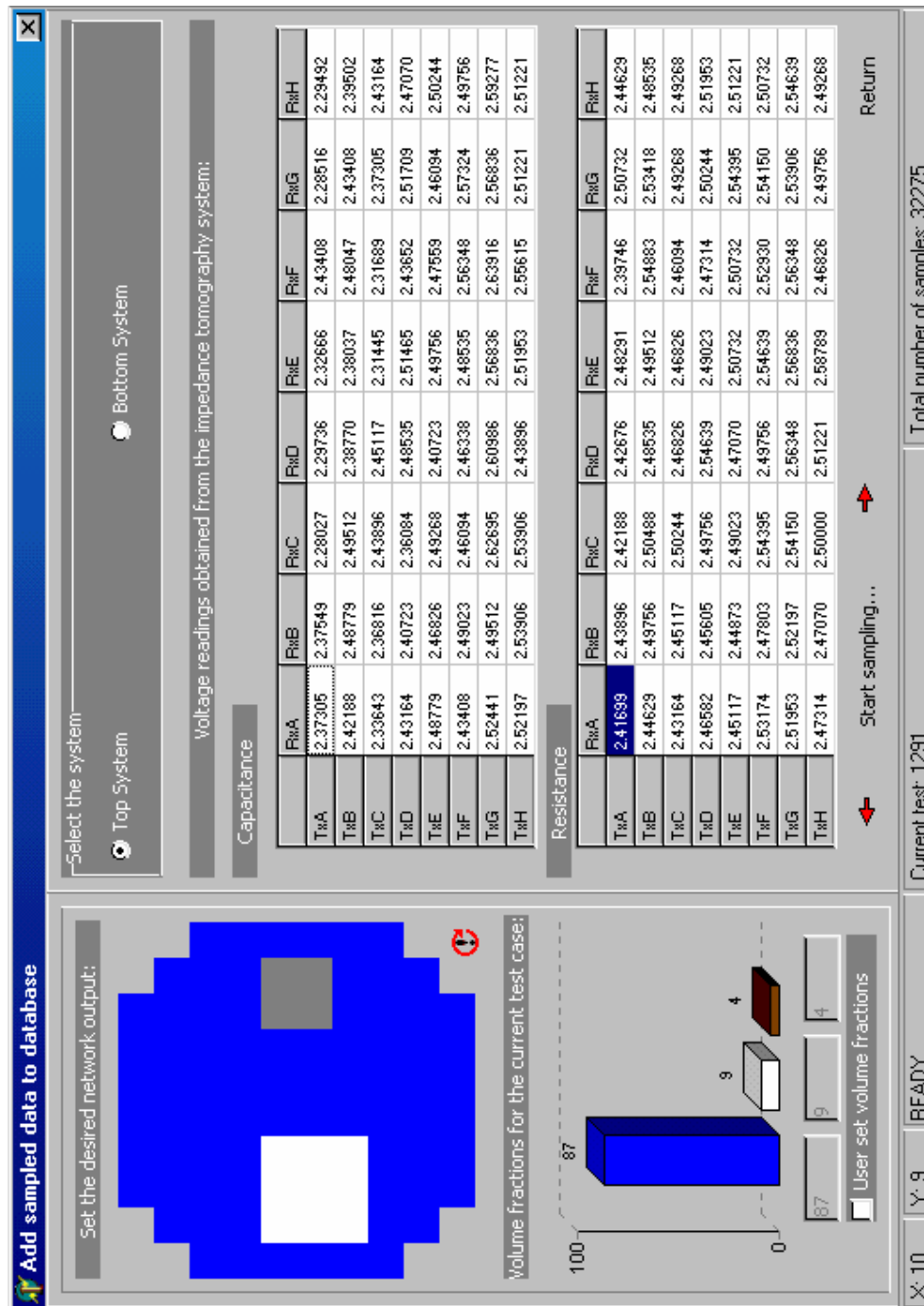
if strtfloat(StepperConfig.AirVelocity.Text)>0 then
    tempstring := inttostr(airposition)+
    inttostr(floor(10*strtfloat(StepperConfig.AirVelocity.Text)))+ 'a'
else
    tempstring := inttostr(gravelposition)+
    inttostr(floor(10*strtfloat(StepperConfig.GravelVelocity.Text)))+ 'g';

filename := 'c:\TestData\dyn'+tempstring+'.txt';
end;
AssignFile(tempfile,filename);
Rewrite(tempfile);
WriteLn(tempfile,RealMain.DesiredFrames.Value);
for row := 1 to RealMain.DesiredFrames.Value do
begin
    DecodeTime(outputstore[row],sampletime,hours,mins,secs,msecs);
    totalmsecs := 1000*(mins*60 + secs)+msecs;
    Write(tempfile,totalmsecs);
    Write(tempfile,row*RealMain.frameinterval:8:4);
    CalculateDistance(row*RealMain.frameinterval,topairposition,
        topgravelposition);
    Write(tempfile,topairposition:8:4);
    Write(tempfile,topgravelposition:8:4);
    for col := 1 to 4 do
    begin
        if totaloutputs = 2 then
            Write(tempfile,outputstore[row],prediction[col]:8:4)
        else
            Write(tempfile,volumestore[row,col]:8:4);
        end;
    for col := 1 to 4 do
        Write(tempfile,desiredvolume[row,col]:8:4);
    Write(tempfile,aircorr[row]:8:4);
    Write(tempfile,gravelcorr[row]:8:4);
    WriteLn(tempfile);
    end;
with RealMain do
begin
    WriteLn(tempfile,'desired frames: '+inttostr(DesiredFrames.Value));
    WriteLn(tempfile,'desired output: '+inttostr(totaloutputs));
    WriteLn(tempfile,'view results: '+RealOnline.Items[RealOnline.ItemIndex]);
    WriteLn(tempfile,'top system test number: '+inttostr(TopTestNum.Value));
    WriteLn(tempfile,'bottom system test number: '+inttostr(BottomTestNum.Value));
    WriteLn(tempfile,'system separation: '+ SystemSepInput.Text);

    WriteLn(tempfile,'flowmeter factor: '+ FlowFactorInput.Text);
    WriteLn(tempfile,'top air threshold: '+ TopAirThresh.Text);
    WriteLn(tempfile,'top gravel threshold: '+ TopGravThresh.Text);
    WriteLn(tempfile,'bottom air threshold: '+ BotAirThresh.Text);
    WriteLn(tempfile,'bottom gravel threshold: '+ BotGravThresh.Text);
    WriteLn(tempfile,'correlation threshold: ' + CorrThreshInp.Text);
    WriteLn(tempfile,'framerate: ' + frmrate.Caption);
    WriteLn(tempfile,'actual air velocity: ' + AirVelActual.Caption);
    WriteLn(tempfile,'actual gravel velocity: ' + GravelVelActual.Caption);
    WriteLn(tempfile,'predicted air velocity: ' + AirVelPredict.Caption);
    WriteLn(tempfile,'predicted gravel velocity: ' + GravelVelPredict.Caption);
end;
WriteLn(tempfile,'flow direction: '+
    StepperConfig.FlowDirSelect.Items[StepperConfig.FlowDirSelect.ItemIndex]);
WriteLn(tempfile,'air bubble diameter: '+ AirDiameter.Text);
WriteLn(tempfile,'air bubble length: '+AirLength.Text);
WriteLn(tempfile,'air bubble position: '+inttostr(airposition));
WriteLn(tempfile,'air bubble start position: '+AirBubbleStartPosition.Text);
WriteLn(tempfile,'gravel bubble diameter: '+GravelDiameter.Text);
WriteLn(tempfile,'gravel bubble length: '+GravelLength.Text);
WriteLn(tempfile,'gravel bubble position: '+inttostr(gravelposition));
WriteLn(tempfile,'gravel bubble start position: '
    +GravelBubbleStartPosition.Text);
WriteLn(tempfile,'upper system top: '+UpperSystemTop.Text);
WriteLn(tempfile,'upper system bottom: '+UpperSystemBottom.Text);
WriteLn(tempfile,'lower system top: '+LowerSystemTop.Text);
WriteLn(tempfile,'lower system bottom: '+LowerSystemBottom.Text);
WriteLn(tempfile,'source of bubble velocity: '
    +VelSource.Items[VelSource.ItemIndex]);
WriteLn(tempfile,'top air volume fraction error: '+TopAirVolError.Caption);
WriteLn(tempfile,'top air bubble delay: '+TopAirDelay.Caption);
WriteLn(tempfile,'top gravel volume fraction error: '+TopGravVolError.Caption);
WriteLn(tempfile,'top gravel bubble delay: '+TopGravDelay.Caption);
WriteLn(tempfile,'bottom air volume fraction error: '+BotAirVolError.Caption);
WriteLn(tempfile,'bottom air bubble delay: '+BotAirDelay.Caption);
WriteLn(tempfile,'bottom gravel volume fraction error: '
    +BotGravVolError.Caption);
WriteLn(tempfile,'bottom gravel bubble delay: '+BotGravDelay.Caption);
CloseFile(tempfile);
end;
end.

```

SCREEN CAPTURES OF SAMPLER PROGRAM



SCREEN CAPTURE OF ADDUNIT.PAS

Continuously sample data from impedance tomography system

Select the system—

☒ Top System
☐ Bottom System

Set recording rate (p/h):

Start recording
Return

Voltage readings obtained from the impedance tomography system:

Capacitance

	RmA	RmB	RmC	RmD	RmE	RmF	RmG	RmH
TmA	2.371	2.380	2.290	2.300	2.327	2.437	2.288	2.297
TmB	2.424	2.500	2.502	2.395	2.395	2.478	2.437	2.397
TmC	2.329	2.371	2.441	2.449	2.310	2.307	2.368	2.424
TmD	2.429	2.405	2.366	2.483	2.522	2.441	2.522	2.473
TmE	2.483	2.461	2.493	2.402	2.500	2.473	2.449	2.490
TmF	2.434	2.480	2.454	2.456	2.480	2.561	2.571	2.495
TmG	2.534	2.498	2.637	2.615	2.578	2.649	2.566	2.595
TmH	2.524	2.537	2.549	2.446	2.520	2.556	2.510	2.510

Resistance

	RmA	RmB	RmC	RmD	RmE	RmF	RmG	RmH
TmA	2.417	2.437	2.422	2.424	2.478	2.395	2.505	2.446
TmB	2.446	2.495	2.505	2.485	2.493	2.554	2.527	2.488
TmC	2.434	2.451	2.505	2.471	2.466	2.461	2.493	2.490
TmD	2.466	2.456	2.498	2.546	2.485	2.471	2.500	2.520
TmE	2.449	2.446	2.490	2.476	2.507	2.507	2.546	2.515
TmF	2.532	2.478	2.544	2.498	2.549	2.524	2.542	2.505
TmG	2.515	2.520	2.539	2.563	2.571	2.563	2.539	2.546
TmH	2.466	2.468	2.490	2.512	2.581	2.466	2.495	2.485

SCREEN CAPTURE OF CURUNIT.PAS

APPENDIX N

SCREEN CAPTURES OF NEURAL NETWORK PROGRAM

Load training and testing databases

Specify the desired reconstruction

☐ Image of vessel cross-section

☒ Volume fraction for different phases

Specify the desired phase

☒ air-gravel-water three-phase mixture

☐ air-water two-phase mixture

☐ gravel-water two-phase mixture

Training database

Specify database name: TrainTomo

First train case to load: 1

Last train case to load: 1290

Number of training versions: 5

Testing database

Specify database name: TestTomo

First test case to load: 1

Last test case to load: 1290

Number of test versions: 5

Loaded 12900 datapoints from the upper system

Load databases...

Pre-process the loaded data


The data will be pre-processed using standardisation. 'Calibrate' performs reconstructions based on the differences between a set of samples and the average voltages for the rig full of water.

☒ Calibrate

Pre-process data...

Return

SCREEN CAPTURE OF LOADUNIT.PAS


Set the network training parameters

General parameters

Maximum number of epochs: 2000

Number of hidden layer neurons: 5

☒ Perform early stopping to prevent over-fitting
☒ Conduct model search

Early stopping parameters

Number of failures before training stops: 2

Stopping condition:
☒ Sum total of volume fraction errors

Network specific parameters

☐ Use gradient descent
☒ Use Resilient back-propagation

Specify the initial delta value: 0.0001
 Specify the maximum delta value: 50
 Specify the performance refresh rate: 1

Model search parameters

Minimum number of hidden layer nodes: 5

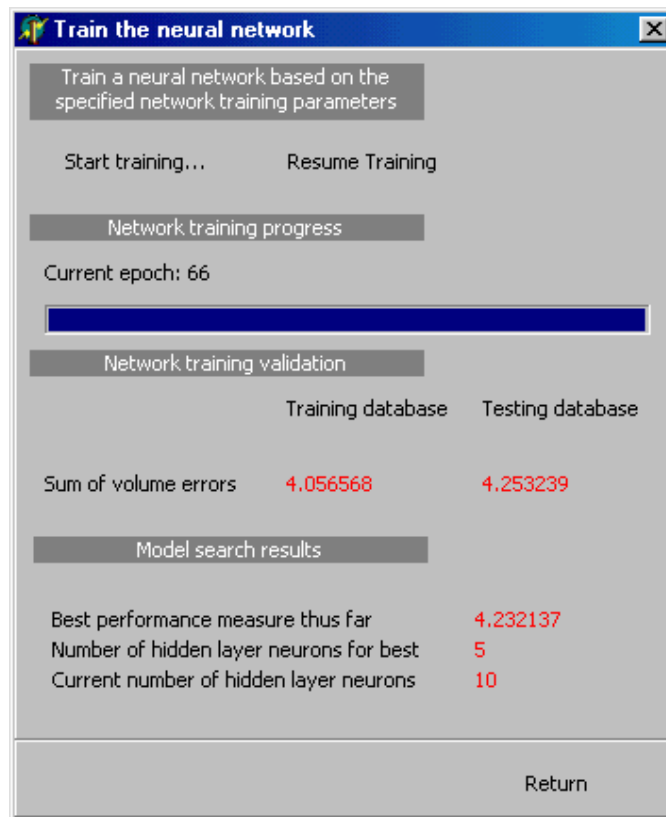
Maximum number of hidden layer nodes: 100

Increment: 5

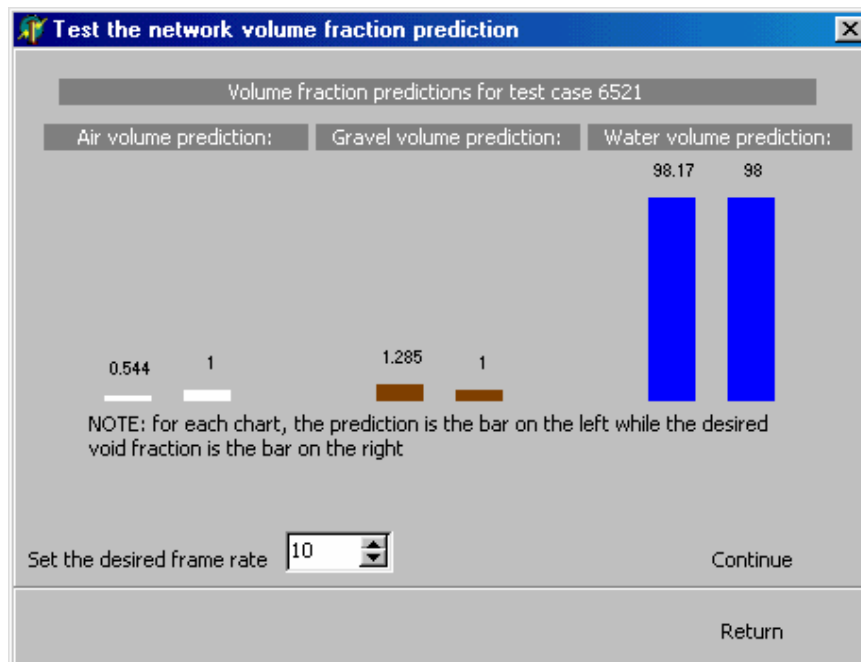
☒ View the training database performance measures

Return

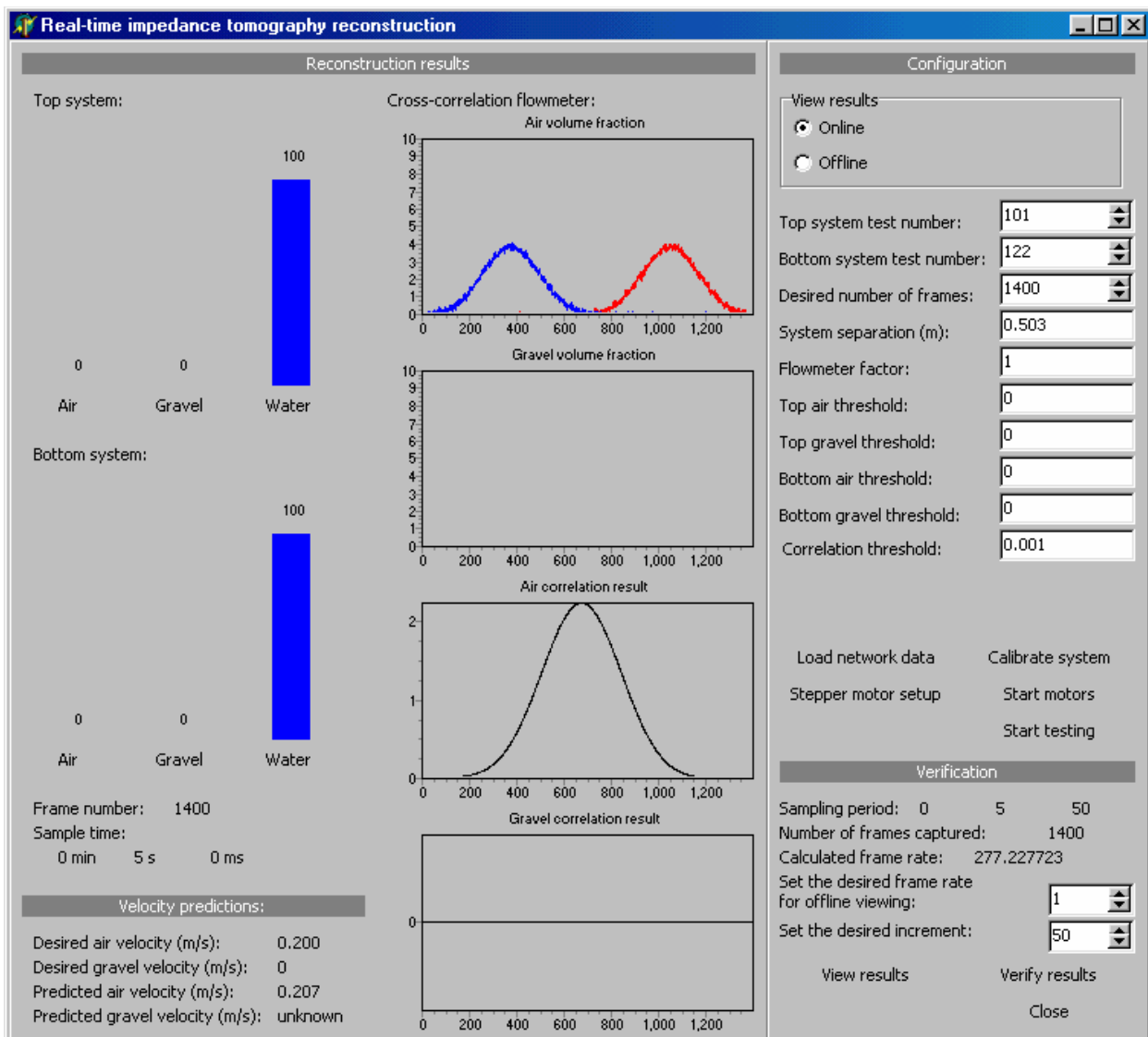
SCREEN CAPTURE OF PARAMUNIT.PAS



SCREEN CAPTURE OF NETWORKUNIT.PAS



SCREEN CAPTURE OF TESTUNIT.PAS



SCREEN CAPTURE OF REALUNIT.PAS

Stepper motor configuration

Overall controller configuration

Flow direction

☐ downward flow
☒ upward flow

Download operating system
Initialise stepper motors
Reset motor limits

Individual motor configuration

Air bubble motor

Drive number

1

Pulley diameter (m)

0.14

Velocity (m/s)

0.2

Distance (m)

1.05

Acceleration (rps/s)

50

Bubble position

☐ inside pulley
☒ outside pulley

Jog air bubble

Additional bubble motor

Drive number

2

Phase

air

Pulley diameter (m)

0.14

Velocity (m/s)

0

Distance (m)

1.05

Acceleration (rps/s)

50

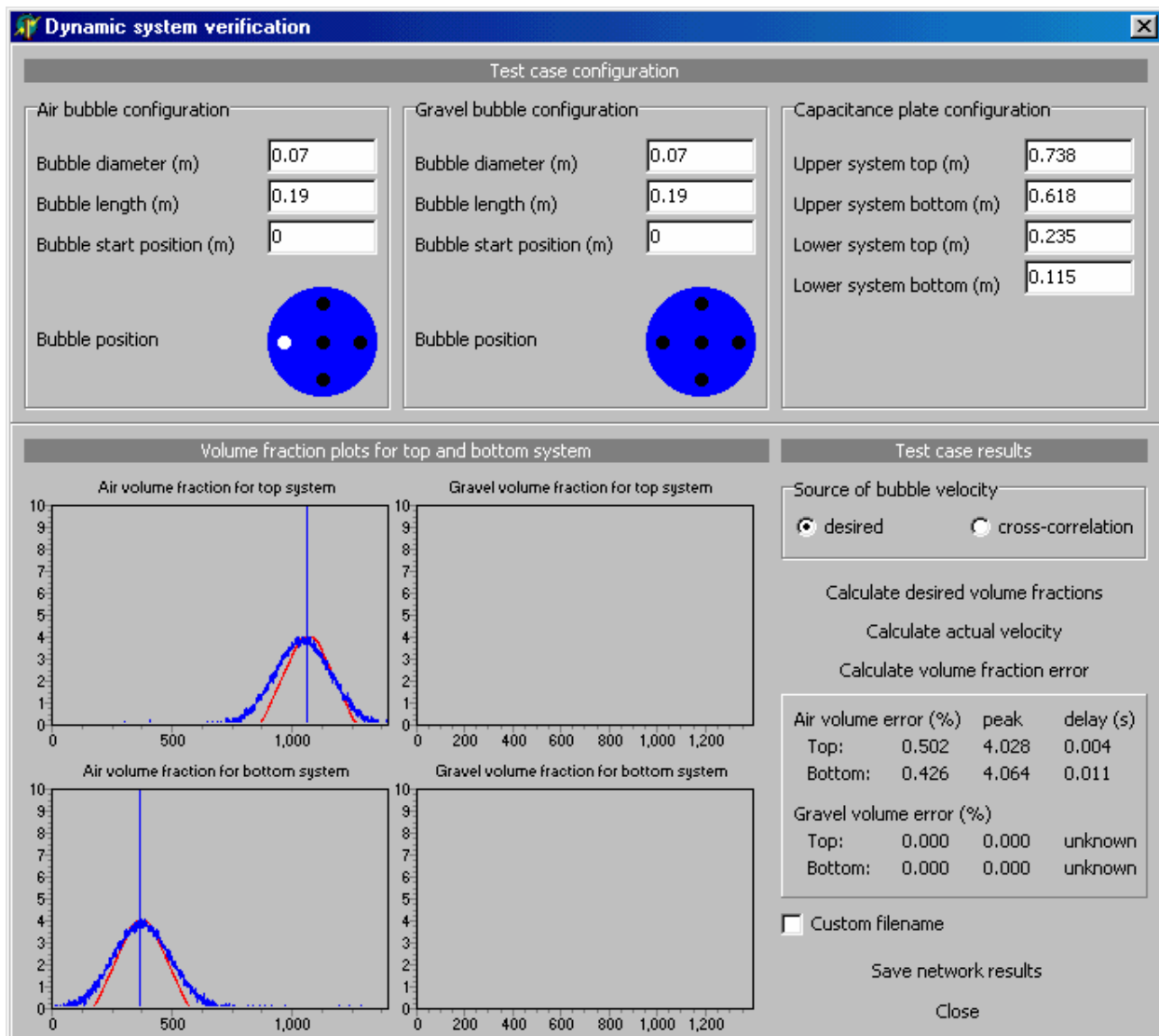
Bubble position

☐ inside pulley
☒ outside pulley

Jog gravel bubble

Close

SCREEN CAPTURE OF STEPCONFIG.PAS



SCREEN CAPTURE OF DYNUNIT.PAS

APPENDIX O
PAPER PUBLISHED IN MEASUREMENT SCIENCE AND TECHNOLOGY
JOURNAL ON AUTHOR'S PREVIOUS RESEARCH